LOCALIZACIÓN GLOBAL PARA ROBOTS MÓVILES BASADA EN SIMULATED ANNEALING

TRABAJO FIN DE GRADO

UNIVERSIDAD CARLOS III DE MADRID

Grado en ingeniería electrónica industrial y automática

Autor: Eduardo Samper Gómez

Tutor: Fernando Martín Monar

Universidad Carlos III de Madrid

AGRADECIMIENTOS

Quiero agradecer a todas las personas con las que he compartido estos años maravillosos de carrera. Una mención especial a mi familia y amigos, presentes en todo momento para apoyarme y ayudarme a continuar.

Resumen

En este trabajo se va a exponer una aplicación del algoritmo Simulated Annealing. Se comenzará dando paso a los marcos de la industria actual, para entender el entorno socioeconómico del mismo. Luego se tratará de dar una base de los conocimientos actuales capaces de resolver problemas de optimización.

Profundizaremos en el Simulated Annealing desde su constitución como evolución de los algoritmos basados en la combinación de los métodos de Monte Carlo y las Cadenas de Márkov. Así como en la evolución que supone al algoritmo de Metrópolis-Hastings. Una vez presentado el conocimiento previo, daremos sentido al nombre de recocido simulado (traducción al español del Simulated Annealing) estudiando cómo se forma este proceso y que influye en su resultado.

Pasaremos a identificar los pasos necesarios para ejecutar el algoritmo y los iremos adaptando a nuestro programa. Así, pasará a formar parte del software de un robot que pretende resolver un problema de localización global. Dicho robot nos plantea un problema de localización en el que, mediante un sensor láser, se desea conocer su posición y orientación real.

Por último, probaremos el software desarrollado en la herramienta de MATLAB en varias pruebas para intentar diagnosticar posibles problemas de ejecución. Dando una solución al problema inicial, y proponiendo posibles mejoras futuras del código.

Abstract

This work will present an application of the Simulated Annealing algorithm. It will begin giving way to the socioeconomic frames in the current industry. Then we will try to give a base, explaining the current knowledge capable of solving optimization problems.

We will delve into the Simulated Annealing since its establishment as an evolution of algorithms based on the combination of Monte Carlo methods and Markov chains. Just as in the evolution that represents of the Metropolis-Hastings algorithm. Once submitted prior knowledge, we give meaning to the name of simulated annealing studying how this process is formed and which variables are influencing its outcome.

We will identify the steps needed to make the algorithm and we will adapt them to our program. Thus, it will become part of the software of a robot that has to solve a problem of global location. Said robot, using a laser sensor, wants to know it's position and actual orientation.

Finally, we will test the software, developed in a tool called MATLAB, in several tests to try to diagnose possible implementation problems. Giving a solution to the initial problem, and proposing possible future improvements.

Estructura del trabajo

En este apartado se va a exponer como se estructura el trabajo completo. Dicho trabajo se compone de diferentes capítulos que se expondrán a continuación.

Estado del arte. En este capítulo se abordarán las soluciones alternativas más significativas para nuestro trabajo. Dando un conocimiento general de los tipos de algoritmos de la optimización combinatoria heurística. Para ir centrándonos en las alternativas del Simulated Annealing, tales como los algoritmos evolutivos.

Simulated Annealing. En este capítulo vamos a asentar las bases de nuestro conocimiento, empezando por exponer los principios en los que se fundamenta el método, que son el método de Monte Carlo y las Cadenas de Márkov. Terminaremos ampliando la información del recocido como tratamiento del acero y usaremos ese conocimiento para establecer los principios del Simulated Annealing.

Localización Global para robots móviles. En este capítulo se explicará el problema al que nos enfrentaremos. Trataremos de poner en situación al lector de las herramientas de las que debemos hacer uso, como los diferentes mapas y funciones, para dar una solución al problema de localización global de un robot.

Solución propuesta. En este capítulo se dará la solución a la que hemos llegado mediante diagramas y código elaborado con la herramienta MATLAB. Se dará una descripción detallada de cómo hemos ido enfrentando los diferentes desafíos del mismo.

Resultados obtenidos. En este capítulo se irán proponiendo ejecuciones del código, con el objetivo de buscar fallos y pulir el mismo hasta su versión final.

Conclusión. En este capítulo se tratará de aportar una serie de mejoras en las que podremos trabajar en un futuro próximo. Además de exponer todas las conclusiones a las que hemos podido llegar tras todo el proceso del trabajo.

Índice

ÍNDICE DE FIGURAS

ÍNDICE DE DIAGRAMAS Y TABLAS

1. INTRODUCCIÓN.

Actualmente vivimos en un mundo de cambio constante, somos los privilegiados de tener mano de obra automatizada capaz de generar trabajos pesados en pocos minutos y con una precisión antes imposible. Esto nos capacita para un cambio constante, una evolución continua, tanto en la técnica, como en el conocimiento. Por ello, considero al ordenador el descubrimiento más importante en la historia de la humanidad. Nos capacita para procesar operaciones de cualquier tipo o dimensión. Lo que ha significado formar parte de la época de mayor número de revoluciones tecnológicas. Hemos vivido cambios tan importantes como el teléfono móvil, y nuestros padres la televisión o internet, y nuestros abuelos el coche. Cada uno de nosotros podemos contar a nuestros hijos lo diferente que era el mundo antes, pero lo más importante es que ellos vivirán la siguiente revolución tecnológica.

Cuando profundizamos más en este ámbito, debemos de hablar de procesadores, robots, o incluso redes de internet de alta velocidad. Todo ello sumado genera la gran industria altamente capacitada que conocemos actualmente. Hablando de manera más específica podemos adoptar el término de industria 4.0.

Industria 4.0 o industria inteligente [1] es la creciente y adecuada digitalización, además de la cooperación y coordinación de todas las unidades productivas. Esto se traduce en plantas industriales inteligentes, basadas en robots con bases de datos, un uso intensivo de internet y una gran comunicación entre los mercados de oferta y demanda. Lo que permite una gran adaptabilidad a las necesidades y procesos de producción, siendo más eficientes en los recursos.

La industria 4.0 también resuelve problemas de ahorro de energía o gestión de los recursos, tanto naturales como humanos. Y es ahí dónde vamos a desarrollar nuestro

proyecto, puesto que en este entorno socioeconómico se enmarca nuestro trabajo. Y es que no podemos entender el mundo sin la robótica industrial.

El ámbito de la robótica, a nivel industrial, puede definirse como la técnica que aplica la información al diseño y empleo de aparatos que, en sustitución de las personas, realizan operaciones o trabajos. [2]

Es ya vital para cualquier fábrica, un manipulador programable. Que ejerza las tareas más complicadas de manera automática. Cada día es más común su presencia en cualquier tipo de sector, y tanto ha crecido su oferta que ya no solo debemos pertenecer el sector profesional o industrial, sino que podemos comprar alguna versión para cocinar, lavar la ropa o realizar por nosotros cualquier tipo de tarea doméstica.

Para poder entender cómo funcionan estas máquinas debemos entender sus procesos y como optimizan las diversas funciones que los componen. Como veremos en los siguientes apartados, los métodos que vamos a introducir y nuestro método propuesto, el SA, son métodos pertenecientes a la optimización combinatoria. Una de las principales funciones que desempeña cualquier robot; resolviendo problemas de optimización será capaz de desempeñar arduas tareas. Nos adentraremos en su propia inteligencia.

Estos algoritmos usarán las bases de los métodos no deterministas y heurísticos, como método de Monte Carlo y de las Cadenas de Márkov. Estas definiciones se presentarán en el siguiente capítulo y son fundamentales para el entendimiento del SA, ya que es una adaptación de las mismas. Una vez entendido todo el fundamento teórico necesario para entender el SA, pasaremos a entender la técnica del recocido que tratará de adaptarse para resolver problemas de optimización en la búsqueda de mínimos globales.

Para poder llevar a cabo dicha adaptación será necesario basarse en un método de Monte Carlo llamado Metrópolis-Hastings que será el punto de partida para nuestro SA. A partir de este punto comenzaremos a hablar de nuestro problema particular y como hemos adoptado una solución basada en SA para resolverlo.

1.1 MOTIVACIÓN.

Desde antes de empezar la carrera ya me fascinaba la tecnología, tenía mucha curiosidad por conocer cómo nacen las ideas que finalmente nos proporcionan esos objetos cotidianos, esos descubrimientos que nos cambian nuestra forma de vivir para siempre.

Como todos los alumnos comenzamos estudiando diversos campos, pero siempre he tenido gran debilidad por el campo de la ingeniería. Al estudiar historia, me imaginaba como debieron de concebir los coetáneos a la revolución industrial ese gran cambio tecnológico que cambiara la vida humana como se concebía hasta la fecha.

Al comenzar la carrera pude focalizarme en el conocimiento de la electrónica, que es el campo que más admiro. Por un lado, también empezaron a interesarme el resto de campos de estudio de la ingeniería. Por otro lado, conseguí acercarme a mi objetivo y ahora tengo una idea más precisa de cómo va evolucionando la tecnología y con ella la vida que conocemos.

Todo me lleva a pensar que el siguiente cambio que nuestros hijos recordaran para siempre es el de la robótica. Ahora tengo la oportunidad de contribuir a este campo, ayudar a la robótica a dar un paso hacia delante, acercarla un poco más a nuestros hogares.

Por este motivo he elegido realizar este trabajo para dar el cierre final al grado. Tomar un contacto directo con el ámbito y aprender aún más como se hacen los robots.

1.2 OBJETIVO.

Un robot autónomo necesita saber su localización para ejecutar múltiples tareas como la navegación o manipulación de objetos. Por lo tanto, uno de los problemas más importantes de un robot autónomo es su localización global LG, que consiste en buscar sus coordenadas en un entorno conocido sin información de su localización inicial. Es posible

distinguir entre dos diferentes sistemas de LG según la fuente de información. Los sistemas de posicionamiento reciben las señales de una fuente externa. Un ejemplo es el conocido sistema de posición global (GPS). Los sistemas de auto-posicionamiento están basados en sensores implementados dentro del robot que se suelen organizar en módulos tales como: módulos de localización basados en rangos láser con los que obtienen escaneos en dos (2D) y tres dimensiones (3D). Este trabajo está incluido en la segunda opción (3D) ya que nuestro robot trabaja en sitios cerrados y la información viene dada por un sensor láser.

Aquí comenzamos el camino que vamos a recorrer a lo largo de este trabajo. La idea principal es solucionar el problema de la LG mediante el método del Simulated Annealing (SA) o recocido simulado.

El SA es un algoritmo que deberá optimizar una función de costes para resolver un problema de LG, además, tendrá una componente poblacional, ya que lo aplicaremos a varios puntos al mismo tiempo. Para poder resolver el problema de LG mediante el método propuesto se ha asimilado las diferentes funciones de coste, los métodos de resolución de dicho problema alternativos y el concepto del SA para aplicarlo a nuestro problema.

Previamente se implementó un algoritmo basado en el método de Evolución Diferencial (DE), que resuelve el problema de LG en entornos 2D y 3D. Estos métodos son técnicas de optimización basadas en la representación de la posición y orientación del robot mediante un conjunto de estimaciones de posibles localizaciones (población) ponderadas por una función de coste. Dicha función ya ha sido programada por varios métodos, pero para facilitar la contratación de resultados, nosotros usaremos la norma L2 como base. El conjunto de soluciones evoluciona en el tiempo para integrar la información del sensor y la información del movimiento del robot.

A continuación, se va a exponer como hemos llegado a proponer este método para resolver nuestro problema.

2. ESTADO DEL ARTE.

Existen diferentes tipos de algoritmos que pueden ser utilizados para resolver el problema de LG, y en este capítulo comentaremos únicamente los más significativos que guardan relación con nuestro trabajo. Un primer grupo formado por los métodos bayesianos, y otro segundo gran grupo formado por los métodos de optimización.

Dentro de los métodos basados en el teorema de Bayes[1] [Fig. 1], propuesto por Thomas Bayes, estos se caracterizan por operar en dos pasos cada uno ejecutado por un grupo de algoritmos. En el primer paso, el movimiento y la información probabilística de la percepción se integran en la función de densidad. La estimación es procesada en un segundo paso en función de un criterio específico, como el punto de densidad máximo o el valor promedio. Estos métodos suelen usar la función de densidad para representar las áreas más factibles. De tal manera que, cuando llegan a converger, toda la distribución se concentra en un área.

En los métodos de optimización, toda la información se usa para generar la función de coste que se minimiza en cada ciclo. La estimación es el elemento con el mejor valor aproximado al final de la ejecución del algoritmo completo.

El primer grupo usa la desviación de la función de pérdidas para lograr una solución y el segundo grupo se basa en la búsqueda estocástica para la mejor solución de la función de pérdidas. A lo largo del primer grupo, podemos incluir los filtros de Kalman[2]. El problema de estos métodos es que no pueden tratar problemas con varias hipótesis. La principal ventaja es su velocidad de computación.

[1] El teorema de Bayes, en la teoría de la probabilidad, es una proposición planteada por el filósofo inglés Thomas Bayes (1702-1761)1 en 1763,2 que expresa la probabilidad condicional de un evento aleatorio A dado B en términos de la distribución de probabilidad condicional del evento B dado A y la distribución de probabilidad marginal de sólo A.

[2] El filtro de Kalman es un algoritmo desarrollado por Rudolf E. Kalman en 1960 que sirve para poder identificar el estado oculto (no medible) de un sistema dinámico lineal con buenos resultados ante ruido blanco aditivo

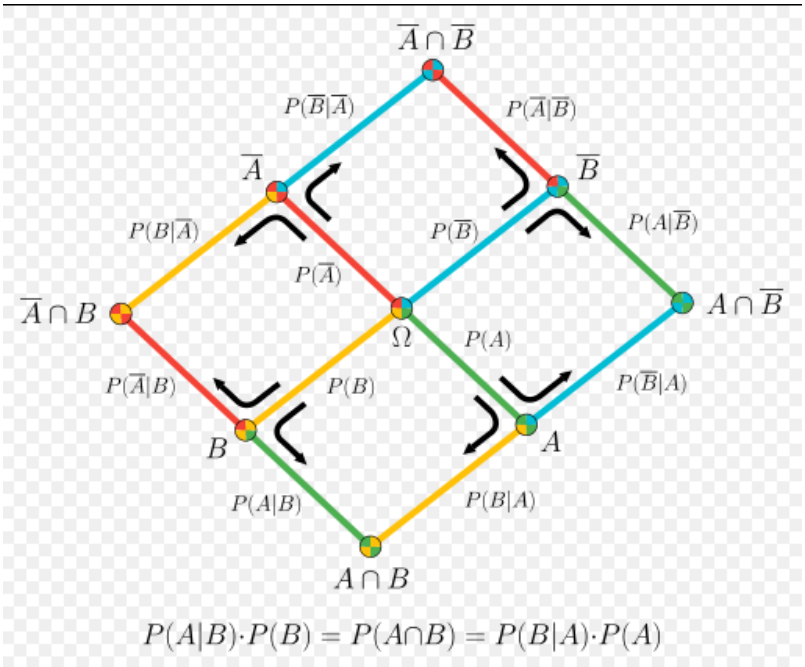


Fig. 1: Ilustración del teorema de Bayes [3].

Los filtros DE pueden ser incluidos en el segundo grupo. Algún método como el propuesto por Vahdat [4] que aplica dos métodos evolutivos (DE y PSO) y compara ambos con el método de Monte Carlo (MC)[3]. Lisowski [5] ha desarrollado el MC basado en DE. Más recientemente, Mirkhania [6] ha implementado un filtro de localización global basado en el algoritmo de Búsqueda Armónica. En nuestro trabajo previo, el algoritmo DE se aplicó a mapas 3D y 2D que será sustituido por nuestra propuesta, el SA.

Los métodos híbridos (filtros multi-hipótesis de Kalman) no son puramente Bayesianos. Sin embargo, con tienen un conjunto de soluciones compuestas por una distribución normal de probabilidad. La generación y eliminación de soluciones están basadas, no solo en la

[3] es un método no determinista y/o estadístico numérico, que se suele usar para aproximar expresiones matemáticas complejas y/o costosas de evaluar con gran exactitud.

distribución de probabilidad, sino también en árboles de decisiones y restricciones geométricas.

Excepto en métodos MC, donde no se minimiza la función de coste, el método previo procesa una función de coste que se apoya en variaciones del error cuadrático entre las observaciones y las medidas estimadas (Norma L2). Estos métodos usan la información de la posición vertical de los elementos del entorno (paredes, columnas, puertas, muebles…) para obtener la función de coste que permite distinguir entre las diferentes localizaciones.

La función de coste más utilizada para la localización de robots móviles es la cuadrática. Sin embargo, hay algunos autores que han considerado diferentes funciones. En nuestro trabajo previo, hemos aplicado la distancia Manhattan (Norma L1) concluyendo que es una aproximación más factible en entornos con objetos dinámicos, personas, etc… Nosotros usaremos la Norma L2 por simplificar y comparar rápidamente los resultados, pero tras probar todos los métodos de la función de coste podemos asegurar que no influyen en nuestro algoritmo.

La Norma L2 ha sido elegida debido a su sencillez. Mide el promedio de los errores al cuadrado, entendiendo el error como la diferencia entre el real y nuestra estimación. Esta sencillez nos permite elaborar unos resultados más rápidos, muy útil para probar el algoritmo hasta su versión final.

2.1 OPTIMIZACIÓN COMBINATORIA.

Los problemas de optimización en los que las variables de decisión son números enteros se denominan problemas de optimización combinatoria. Podemos considerar la optimización combinatoria como una rama matemática que se aplica a la investigación de operaciones, algoritmos y complejidad computacional. Podemos abarcar campos como el de la inteligencia artificial o la robótica.

Los algoritmos que pertenecen a este tipo de optimización se caracterizan por resolver problemas explorando el espacio de posibles soluciones. Reduciendo el tamaño del conjunto de soluciones y buscando la solución óptima de manera eficiente. Como es nuestro caso, debemos hallar una solución en un conjunto finito de soluciones viables. Por otro lado, resulta imposible enumerar dicho conjunto.

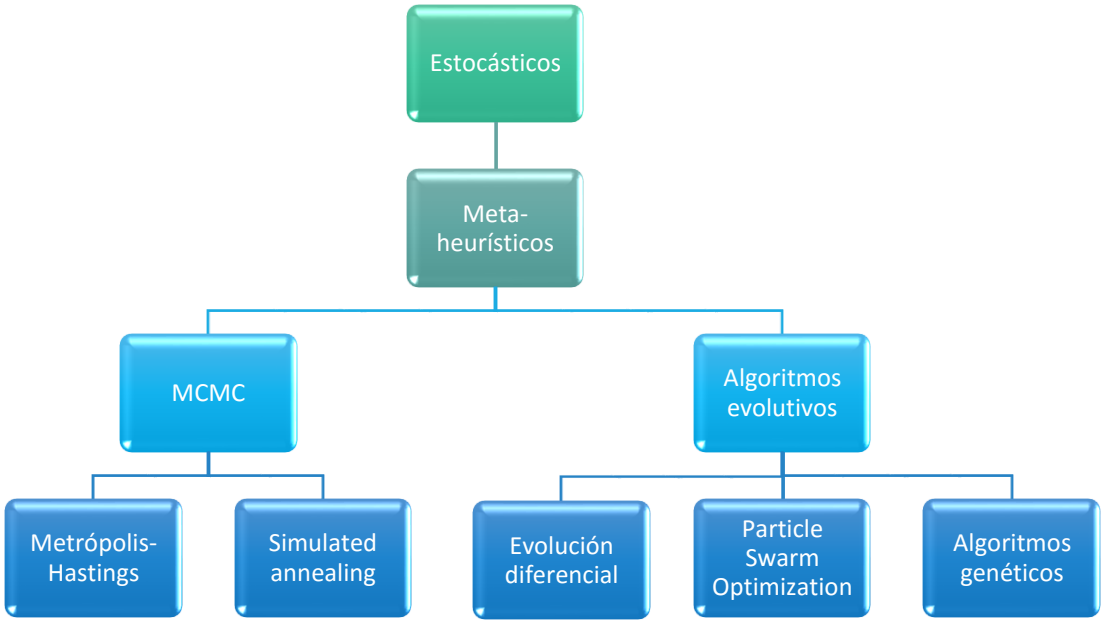


Fig. 2: Esquema de organización de los diferentes métodos de optimización que estudiaremos a lo largo de este trabajo

En los métodos de optimización, toda la información es integrada para generar la función de coste que es minimizada en cada ciclo de percepción del movimiento. La estimación es el elemento con el mejor valor de aproximación.

En los métodos de optimización, hay dos objetivos fundamentales: encontrar algoritmos con buenos tiempos de ejecución y buenas soluciones. Hay algoritmos que abandonan uno o ambos objetivos para encontrar una buena solución. A estos algoritmos

se les llama algoritmos de optimización heurística y hablaremos sobre ellos en el siguiente apartado.

La optimización combinatoria tiene sus raíces en algunos de los problemas económicos: la planificación de las operaciones, el uso eficiente de los recursos, la gestión de las mismas, etc.

Como veremos en los siguientes apartados, los métodos que vamos a introducir y nuestro método propuesto, el SA, son métodos pertenecientes a la optimización combinatoria.

Usaran las bases de los métodos no deterministas heurísticos como método de Monte Carlo y de las Cadenas de Márkov. Estas definiciones se presentarán en el siguiente capítulo y son fundamentales para el entendimiento del SA ya que es una adaptación de las mismas.

2.2 MÉTODOS ESTOCÁSTICOS

Un sistema se comporta de manera estocástica o no determinista[4] cuando el siguiente estado de dicho sistema depende de estados anteriores y también de variables aleatorias. Este término se aplica a los algoritmos y modelos en los que los estados o eventos cambian con el paso del tiempo. Actualmente los más utilizados son los meta-heurísticos como en las tecnologías basadas en métodos evolutivos. Estos métodos, junto con nuestro SA, forman parte de la optimización heurística. Este sub-apartado resulta muy interesante e imprescindible para entender nuestro trabajo, por lo que pasamos a describirlo a continuación.

[4] Palabra clave: no determinista o estocástico. Se usará a lo largo de todo el trabajo.

2.3 LA OPTIMIZACIÓN HEURÍSTICA

Como ya hemos introducido anteriormente, la optimización heurística comprende aquellos algoritmos que pueden abandonar uno de los dos objetivos principales en computación: tiempo de ejecución o solución óptima.

A modo de ejemplo, supongamos una función f objetivo o multi-objetivo arbitraria. f(x) representa la evaluación de dicha función de acuerdo con x. La x representa la solución en el espacio de soluciones factible X perteneciente al conjunto de soluciones posibles Ω.

$$f(x), \quad x \in X \in \Omega$$

Para poder resolver este tipo de problemas podemos recurrir a algoritmos de resolución exacta. Pero en la mayoría de casos reales, el número de variables que debemos usar es muy alto y resulta inviable usar este tipo de algoritmos. Del mismo modo, deberíamos contar con una enumeración implícita, planos de corte, etc.

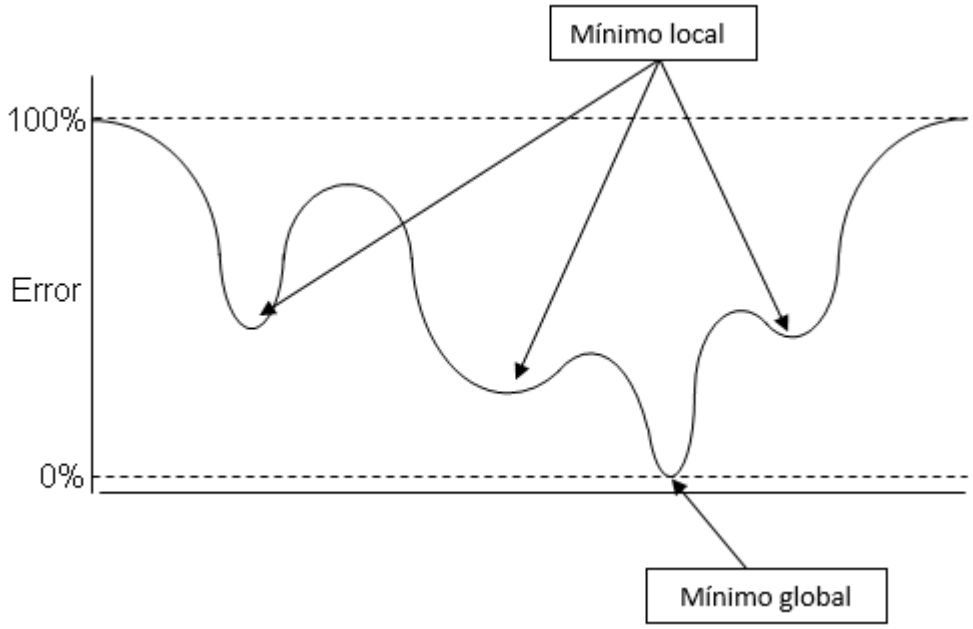


Fig. 3: Ejemplo de función con múltiples máximos locales.

Por otro lado, si optamos por métodos de resolución aproximados podremos contar con la aportación de soluciones satisfactorias en un tiempo de ejecución o resolución razonable.

Empezaremos por el concepto de optimización heurística de búsqueda local. Nos situamos frente a un problema dónde no conocemos el número exacto, pero sabemos que existes numerosos máximos locales [Fig. 3].

Para ello partiremos de una solución aleatoria, que se podrá modificar ligeramente para movernos a una solución mejor mediante una regla determinada.

Este modo de proceder definiría un entorno o vecindario, dicho de otra manera, un conjunto de soluciones a las que se podría llegar mediante un movimiento [Fig. 4].

Como ya hemos planteado, si elegimos un nuevo vecino partiendo siempre de la mejor solución anterior, podríamos caer en un óptimo local y nuestro problema no se podría resolver convenientemente. Por lo tanto, el óptimo local puede encontrarse en un entorno cercano y debemos aceptar soluciones peores para poder alcanzarlo como se muestra gráficamente en [Fig. 5] Esta última consecuencia es la principal característica de los métodos heurísticos.

Dichos métodos se focalizan en minimizar o maximizar la posible solución dentro de este conjunto de soluciones en cada iteración. Como no se puede asegurar un cruce entre la mejor solución ofrecida y la real del 100%, habitualmente estos algoritmos llevan un criterio de tolerancia o parada.

Cuando alcance dicho limite la función debe parar obedeciendo al criterio que, en cada problema, sea oportuno. Este límite suele proponerse de dos maneras diferentes: Cuando tras cierto número de intentos no se consiga mejorar más la solución ofrecida. O bien cuando alcanza una fracción de la condición inicial, es decir, se acerque lo suficiente a la solución óptima.

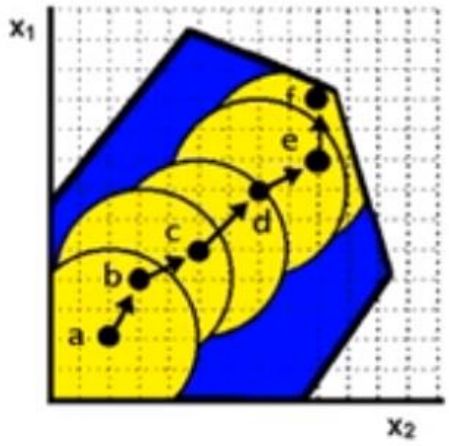


Fig. 4 : Ejemplo gráfico del procedimiento heurístico. [7]

Los métodos heurísticos poseen las siguientes desventajas:

- Los métodos heurísticos habitualmente, debido a su tipo de búsqueda, nos pueden conducir a soluciones erróneas.
- Algunas heurísticas se pueden contradecir al aplicarse al mismo problema.
- Las soluciones óptimas dadas por la heurística pueden llevan a una búsqueda poco exhaustiva.

Por el contrario, poseen las siguientes ventajas:

- Menor tiempo de procesamiento.
- Resultados generalmente aceptables, en el mundo real no suelen darse los casos de errores menos comunes.

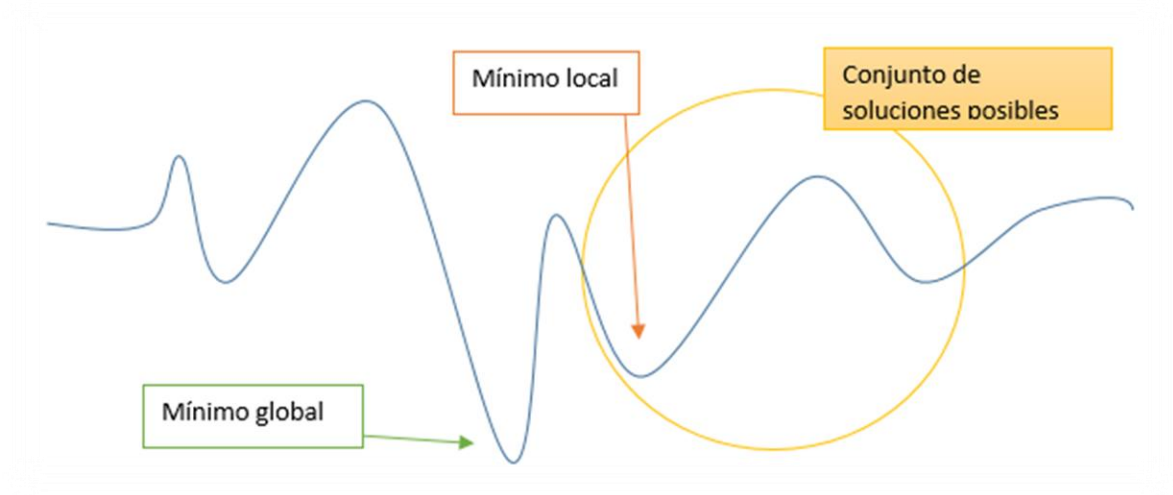


Fig. 5 : Ejemplo del problema al elegir la siguiente solución partiendo siempre de la anterior.

2.4 ALGOLRITMOS EVOLUTIVOS.

Recordamos que en el esquema de la Fig. 2 podemos ver el esquema de ordenación de los diferentes métodos que estudiaremos a continuación, para entender la estructura que seguimos al presentar nuestro trabajo.

Podríamos incluir muchos más, pero presentaremos los más significativos de cara a plantear nuestro trabajo. Estos tres ejemplos serán suficientes para representar el espectro de algoritmos alternativos que surgen ante estos problemas.

2.4.1 Evolución diferencial.

Ya conocemos que la evolución diferencial (DE) es un método de optimización que desciende de la rama de algoritmos evolutivos. Como su nombre nos indica, la DE comprende una población de soluciones posibles que se combinan para evolucionar hacia nuevos candidatos que, según la función objetivo, lidiarán con los anteriores su puesto en la población. Originando una población superior o igual en cada intento. Como ya podréis deducir, el algoritmo consta de cuatro pasos básicos que serán:

1. Generación de población inicial aleatoria.
2. Mutación de tres vectores aleatorios de la población, estos se construyen de acuerdo a la siguiente ecuación:

$$n_p^g = \ x_c + F.(x_a - x_b)$$

Siendo p un número comprendido entre 1 y el número de vectores aleatorios mutados o ruidosos. Y F es el gradiente o control de mutación comprendido entre 0 y 2.

3. Recombinación aleatoria con los vectores anteriores, obteniendo vectores de prueba en función de un parámetro de recombinación comprendido entre 0 y 1. Sigue la siguiente ecuación:

$$t_{p,m}^g = \begin{cases} n_{p,m}^g \ si \ rand(0,1) < GR \\ x_{p,m}^g \ en \ el \ caso \ contrario \end{cases}$$

4. Selección de los mejores vectores en función de la función de coste (fit)

$$x_p^{g+1} = \begin{cases} t_p^g \ si \ fit(t_p^g) > fit(x_p^g) \\ x_p^g \ en \ el \ caso \ contrario \end{cases}$$

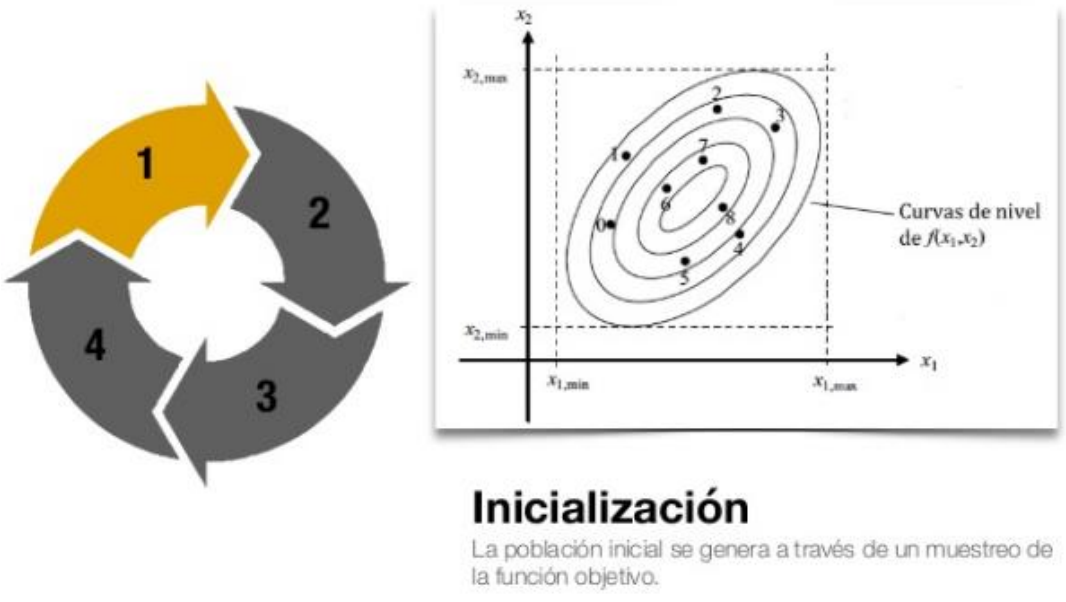


Fig. 6: Ejemplo ilustrativo de los 4 pasos del método de DE. [26]

2.4.2 Algoritmos genéticos.

Comenzó John Holland en 1692 con la investigación de lo que ahora conocemos como algoritmos genéticos. Constituyen un método de búsqueda, pero en este caso, imitando la evolución biológica. Donde los individuos más capaces de la población son reproducidos y mutados para generar la siguiente generación. Este proceso obedece el siguiente esquema:

1. Generar una población inicial, idealmente debe ser de forma aleatoria para cubrir todas las soluciones posibles.
2. Evaluar cada individuo de manera que se pueda distinguir los más capaces.
3. Reproducir los cromosomas, dando más peso a los más aptos.
4. Mutar un gen de un individuo generado con cierta probabilidad.
5. Ordenar la nueva población.

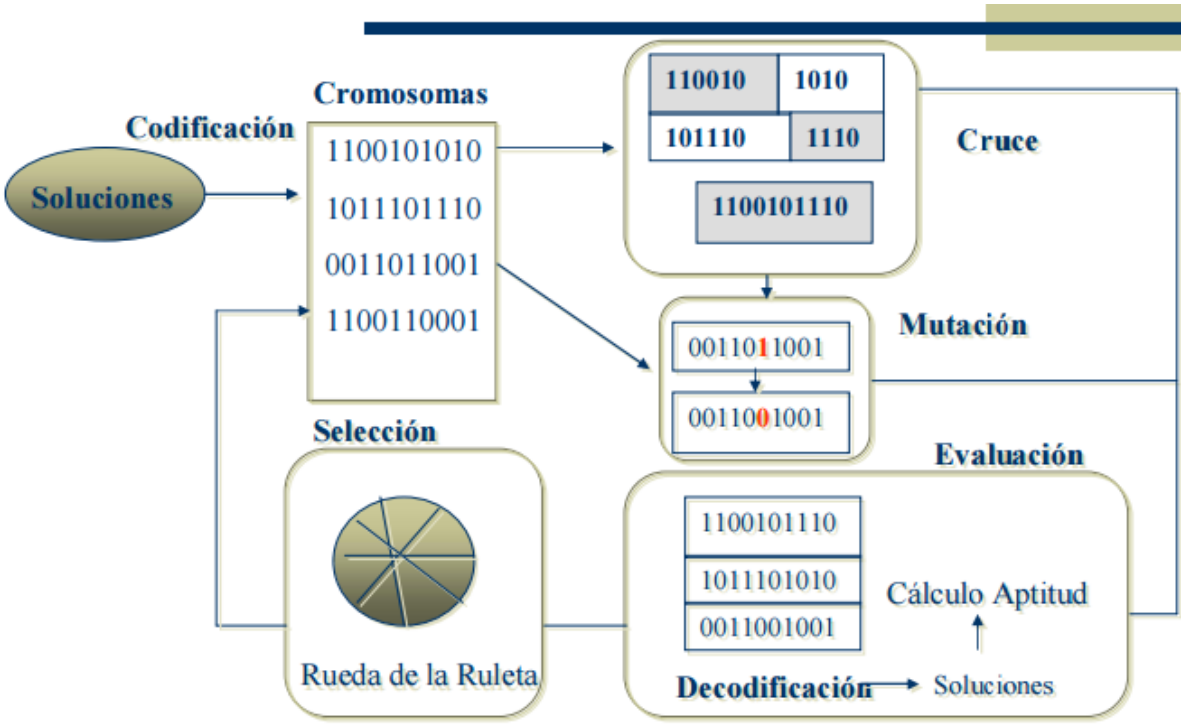


Fig. 7: Esquema de un algoritmo genético básico [8].

Estos algoritmos poseen tres parámetros importantes que influyen en el rendimiento y eficacia del mismo, y son:

- Tamaño de la población. Por un lado, un número bajo conlleva pocas mutaciones y una resolución poco óptima. Un número excesivo no aportará mejores soluciones y además disminuirá la velocidad de resolución.
- Probabilidad de cruce. Una baja probabilidad indica que los hijos serán copias idénticas de los padres sin mejoras. Una alta probabilidad impedirá seleccionar los mejores cromosomas y será totalmente por cruce.
- Probabilidad de mutación. Nos indica con qué frecuencia mutan los cromosomas. Si no muta los descendentes no cambiarán tras la reproducción, pero de ser demasiada la población empobrecería rápidamente.

Las principales ventajas que podemos conseguir con estos algoritmos son:

- Operan varias soluciones a la vez, y pueden abandonar las operaciones menos óptimas al mismo tiempo que avanzan las mejores soluciones.
- Pueden evitar el estancamiento en máximos locales.
- Pueden manejar varios parámetros y seguir varios objetivos al mismo tiempo.
- No necesitan variables especificas del problema, ya que estudian la efectividad de mu mejora aleatoria con la función e coste.

Las principales desventajas que podemos padecer en su uso son:

- Debemos definir el problema como listas de números u otra alternativa robusta frente a cambios aleatorios.
- Pueden tardar mucho en converger o no llegar a hacerlo en función de los parámetros antes citados.
- Si un individuo destaca mucho rápidamente puede reproducirse excesivamente rápido y eclipsar soluciones óptimas de forma prematura.

Por lo que podemos concluir que este tipo de algoritmos supone una solución eficiente a problemas complejos pudiendo operar en paralelo diversos objetivos, pero no podemos asegurar una efectividad plena. Debemos procurar una buena diversidad de la población, por ejemplo, mutando de manera correcta los individuos.

2.4.3 Particle Swarm Optimization.

Conocida comúnmente por sus siglas en ingles PSO, u optimización por enjambre de partículas fue desarrollada por el Dr. Eberhart y el Dr. Kennedy en 1995.

Está inspirada en el comportamiento de las abejas en sociedad para resolver problemas de optimización. Similar a los algoritmos genéticos basados en poblaciones, el PSO presenta los siguientes pasos:

1. Un enjambre se dispone, de manera aleatoria, a buscar la solución en N dimensiones [Fig. 8].

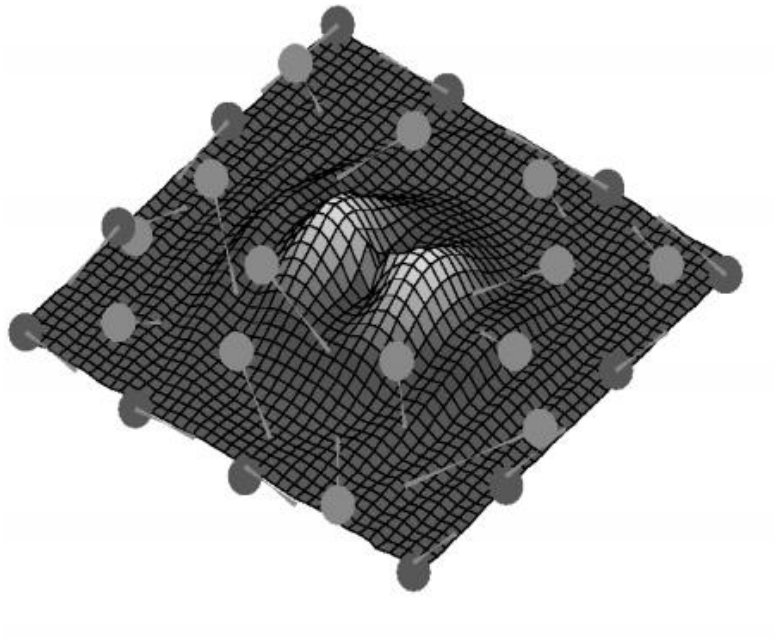


Fig. 8: Ejemplo de partículas moviéndose aleatoriamente en el estado inicial [9].

2. Sólo consideramos una única solución en cada momento.
3. Las partículas desconocen la solución, pero saben el espacio que dista de ellas en cada bucle.
4. Las partículas conocen el espacio existente entre el resto de partículas y ellas mismas.
5. Las partículas deben seguir a aquella que se encuentre más cerca de la solución.

Se diferencian de los anteriores métodos en que no consta de generaciones que mutan o evolucionan, sino que las partículas se mueven en el espacio guiadas por las partículas óptimas actuales.

Para encontrar la partícula optima, que deben seguir las demás, se pueden distinguir dos ámbitos. Uno local, interactuando entre las partículas cercanas a su ubicación. Otra global organizándose en función del resto de partículas en todo el espacio. Dado que se suele caer en mínimos locales cuando hacemos uso del ámbito global, se pueden combinar ambos para obtener más rapidez mediante el método global para posteriormente hacer un ajuste más fino con el local. Obteniendo equilibrio entre precisión y rapidez.

Al igual que en los métodos anteriores, hay varios parámetros que hay que controlar para obtener un buen rendimiento y eficacia del algoritmo. El número de partículas debe rondar entre 10 y 40. Siendo diez suficientes, si aumentamos el número de partículas nos aseguraremos mejores soluciones y peores tiempos de ejecución. Por otro lado, si el espacio de búsqueda es demasiado amplio deberemos aumentar el número de partículas o no podremos concluir que la solución sea optima y no caiga en un máximo local.

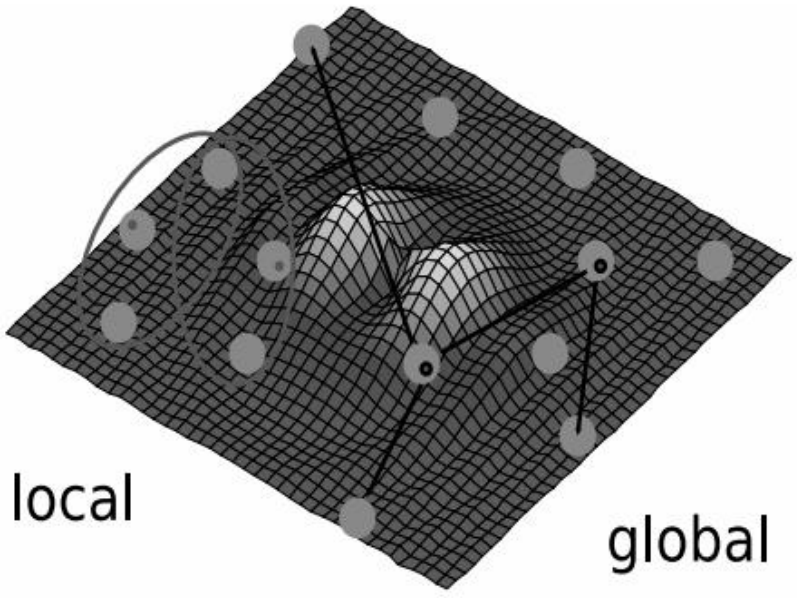


Fig. 9: Ejemplos de ambos métodos de búsqueda del óptimo, social y global [9].

A modo de cierre, voy a presentar un ejemplo muy gráfico de la diferencia de los dos primeros métodos, donde tenemos padres de los que descienden las nuevas generaciones (indicado con la letra “a” en la parte superior de la imagen). Y de cómo las partículas evolucionan a una nueva posición (indicado con la letra “b” en la parte inferior de la imagen). Para poder entender y comparar los tres métodos de manera rápida y clara:

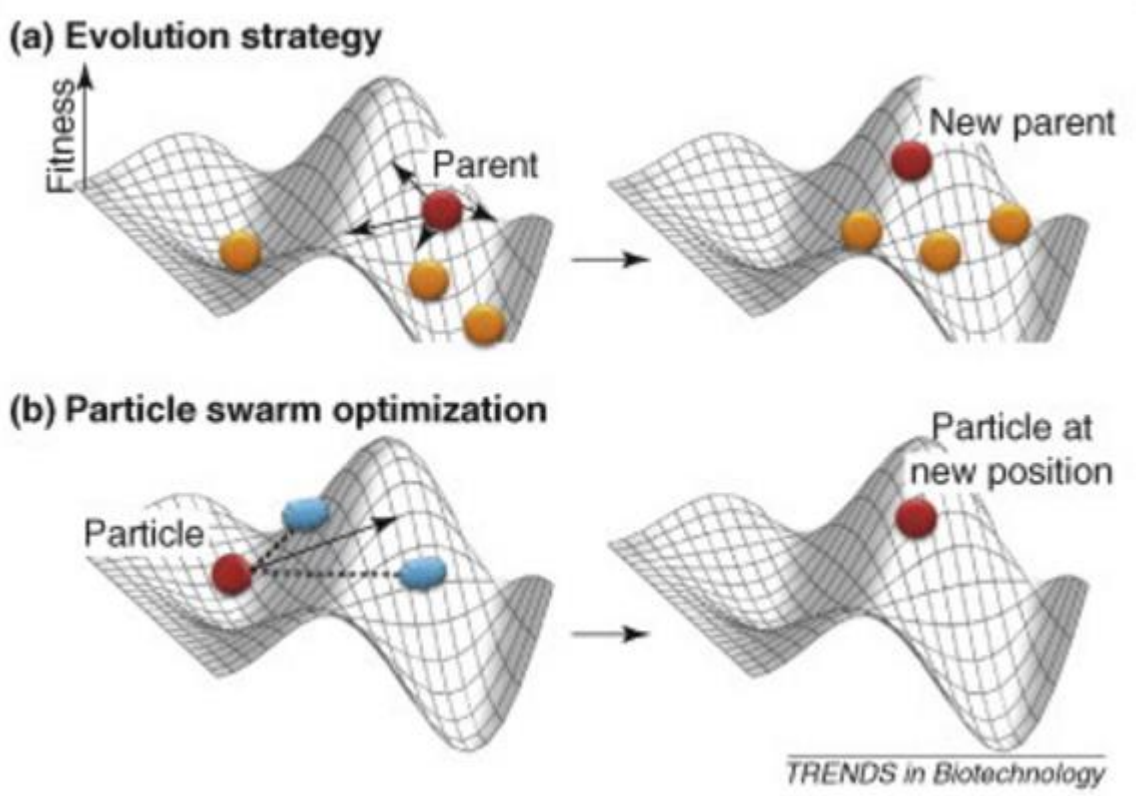


Fig. 10: Ejemplo gráfico de los métodos evolutivos [10].

Mientras que en el a) creamos nuevos padres que mejoran a la población anterior y la acercan a la solución óptima. En el b) movemos las partículas hacía la posición óptima en función del resto de partículas que estén mejor situadas que la candidata al movimiento.

3. SIMULATED ANNEALING

El SA es un método probabilístico propuesto por Kirkpatrick, Gelett y Vecchi en 1983 [11] y dos años más tarde por Cerny [12]. Es un algoritmo de búsqueda de la variante heurística para problemas de optimización. El propósito del algoritmo viene dado por la búsqueda de una buena aproximación a un valor óptimo global. Este método busca un mínimo global de una función de costes que puede poseer varios mínimos locales.

Se basa en la emulación del fenómeno físico llamado recocido donde un sólido se enfría lentamente hasta que, comenzando de manera aleatoria, su estructura se cristaliza. Este momento viene dado cuando se presenta un mínimo de energía de configuración.

3.1 FUNDAMENTOS TEÓRICOS

El SA es un algoritmo de búsqueda de un mínimo global perteneciente a la rama heurística para problemas de optimización.

Usa una adaptación del algoritmo de Metrópolis-Hastings (MH), un algoritmo basado en la aplicación del método de Monte Carlo y las Cadenas de Márkov (MCMC). Ambos métodos son totalmente complementarios, lo que entenderemos a continuación. Pues empezaremos introduciendo estos dos conceptos que formarán los pilares en los que se asientan el SA y el MH.

Si observamos el esquema planteado en Fig. 2, nos vamos adentrando en los métodos de MCMC. Por ello empezaremos introduciendo los fundamentos teóricos que nos servirán para elaborar la adaptación de la técnica del recocido a problemas de optimización.

3.1.1 MÉTODO DE MONTE CARLO

El Método de Monte Carlo (MC) es el primer pilar básico del SA, basa su importancia en la búsqueda de una aproximación de expresiones matemáticas complejas, difíciles de resolver por métodos analíticos, que obedecen a variables aleatorias o pueden asociarse a un modelo probabilístico. Debe su nombre al Casino de Monte Carlo del Principado de Mónaco, por su fama de ciudad experta en juegos de azar, y en especial a la ruleta como método para generar números aleatorios.

Este método se comenzó a usar en la investigación de la bomba atómica en la Segunda Guerra Mundial por parte de EE.UU. Y se basaba en la simulación de problemas de hidrodinámica propios de la probabilidad, específicamente de la difusión de neutrones en el material de fisión. El cual se comporta de manera aleatoria.

El error generado al aplicar estos métodos suele rondar el $\frac{1}{\sqrt{N}}$, siendo N el número de intentos. Dicho de otra manera, al aumentar la precisión en una cifra decimal, implica realizar 100 intentos más.

Cuando hablamos de MC nos referimos a un proceso estocástico numérico, o una concatenación de estados que evolucionan aleatoriamente. No hay una manera exacta de definir el método por tanto no podemos aportar una serie de pasos precisos. Propondremos por tanto el patrón que se suele seguir para generar un método de estas condiciones:

1. Definir el dominio.
2. Generar puntos aleatorios dentro del dominio de manera aleatoria.
3. Usar una técnica determinista para analizar los datos.
4. Observar los resultados.

Vamos a elaborar un ejemplo para ver este proceso:

1. Dibujaremos un circulo dentro de un cuadrado, de lado dos unidades (con valores entre [-1,1].
2. Generaremos puntos aleatorios dentro del cuadrado.
3. Contaremos los números que hay dentro del círculo, por un lado, y por otro lado el total. Como hemos diseñado el círculo, deberemos de contar los de un cuadrante y multiplicarlo por 4 de manera que el recuento sea más cómodo.
4. La relación entre ambos totales será la estimación del área del círculo, como hemos propuesto el número pi.

Vamos a aplicar el ejemplo en MATLAB, y observaremos los distintos resultados al aumentar los intentos, o en nuestro caso los puntos generados en el código (Véase anexo 0).

El objetivo que nos vamos a plantear es conseguir aproximar los dos primeros decimales y comparar los resultados en tres ejemplos de, respectivamente, 1000, 5000 y 10000 puntos.

Debemos tener en cuenta que si los puntos no se distribuyen de manera aleatoria y uniformemente distribuida nuestra aproximación no será lo suficientemente fiable, y por tanto no será correcta. Por otro lado, el número de puntos es muy importante. Como hemos aproximado anteriormente si aumentamos el número de puntos ganaremos precisión. Por lo tanto, un número de puntos excesivamente bajo nos aportarán una solución muy pobre y no fiable, pero un excesivo número de puntos nos aportará una solución altamente fiable y precisa, pero nos llevará un gran número de iteraciones y tiempo.

El uso de métodos de MC requiere por consiguiente una gran cantidad de puntos o números aleatorios ya que la aproximación se incrementa de manera directamente proporcional a dicha cantidad.

Vamos a pasar a describir los resultados de nuestro pequeño experimento utilizando el método de MC para aproximar el valor del número pi.

Cálculo de pi usando 1000 puntos.

Tabla 1: Resultados del cálculo de pi usando 1000 puntos.

Name	Value
ans	3.1416
J	*1000x2 double*
k	*1000x1 logical*
m	773
n	1000
piestimate	3.0920

Podemos concluir que el intento no ha utilizado suficientes puntos, ya que el error es de menos de 0.05 como podemos ver en Tabla 1. Lo estimamos insuficiente ya que no cumple con la aproximación de los dos primeros decimales que hemos estimado como objetivo. Podemos ver el número de aciertos, los puntos que han acertado dentro del círculo, de manera gráfica en la Fig. 11.

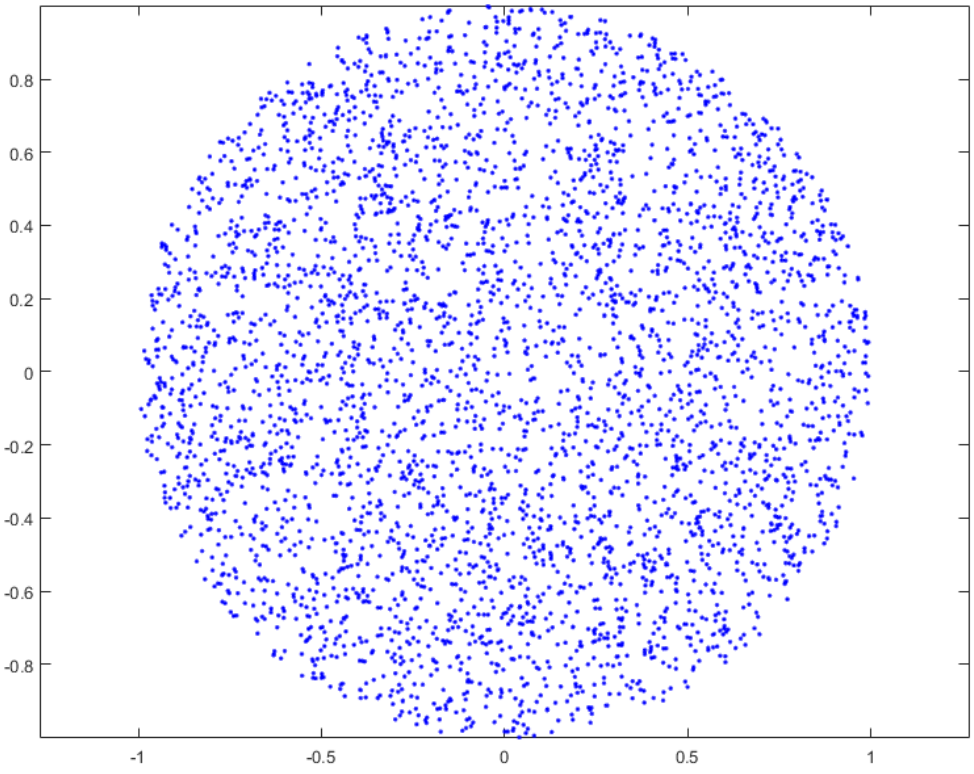


Fig. 11: Estimación 1 del número pi en MATLAB mediante el método MC. .

Calculo de pi usando 5000 puntos.

Tabla 2: Resultados del cálculo de pi usando 5000 puntos.

Name	Value
ans	3.1416
J	*5000x2 double*
k	*5000x1 logical*
m	3971
n	5000
piestimate	3.1768

Podemos concluir que el intento no ha utilizado suficientes puntos, ya que el error es de menos de 0.03 como podemos ver en la Tabla 2. Con el mismo criterio que el caso anterior, lo debemos estimar insuficiente ya que no cumple con la aproximación de los dos primeros decimales que hemos estimado como objetivo. Podemos ver el número de aciertos de manera gráfica en la Fig. 12.

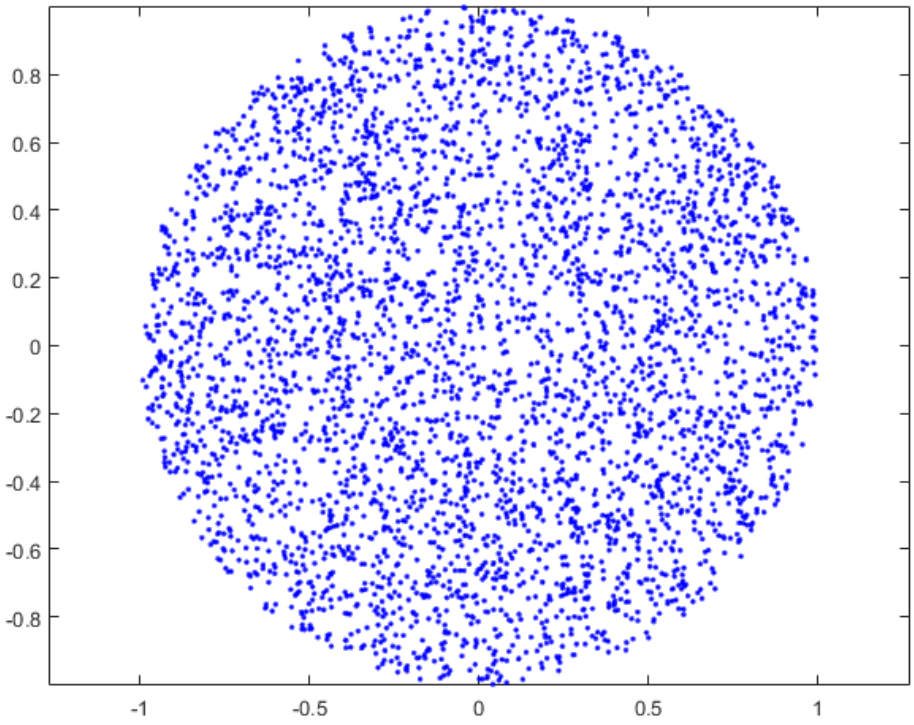


Fig. 12: Estimación 2 del número pi en MATLAB mediante el método MC.

Calculo de pi usando 50000 puntos.

Tabla 3: Resultados del cálculo de pi usando 50000 puntos.

Name	Value
ans	3.1416
J	*50000x2 double*
k	*50000x1 logical*
m	39210
n	50000
piestimate	3.1368

Podemos concluir que el intento no ha utilizado suficientes puntos, ya que el error es de menos de 4,8x10-3 como podemos ver en la Tabla 3. En este caso, lo estimamos suficiente ya que cumple con la aproximación de los dos primeros decimales que hemos estimado como objetivo. Podemos consultar el número de aciertos de manera gráfica en la Fig. 13.

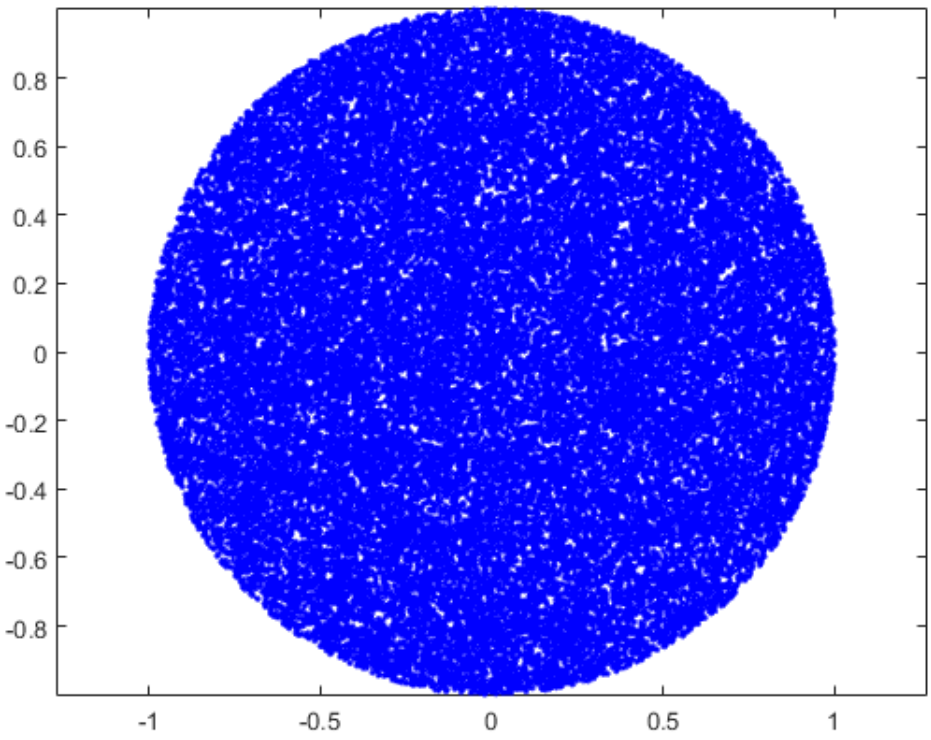


Fig. 13 Estimación 3 del número pi en MATLAB mediante el método MC.

3.1.2 CADENA DE MÁRKOV

Debe el nombre al matemático que propuso dicho método en 1907, Andréi Márkov.

La cadena de Márkov (CM) es un método estocástico cuya característica principal viene dada por la probabilidad de ocurra un evento concreto en función del evento anterior. Cuando solamente depende del evento inmediato que le precede, hablamos de la propiedad de Márkov.

Si se conoce todos los eventos del sistema hasta el actual podemos conocer la probabilidad de su estado futuro, obteniendo toda la información de su estado actual.

El término cadena de Márkov se aplica a una concatenación de variables aleatorias X_1, X_2, X_3, ... X_n. Si la distribución del siguiente estado o X_{n+1} en eventos anteriores es función de únicamente su estado anterior X_n. Entonces:

$$P\,(X_{n+1} = x_{n+1} | X_n = x_n, X_{n-1} = x_{n-1}, \ldots, X_1 = x_1) = P\,(X_{n+1} = x_{n+1} | X_n = x_n)$$

En dichas cadenas, cuando la probabilidad no depende del instante en el que evaluamos n obtendremos:

$$P\,(Xn = j | Xn\text{-}1 = i)$$

Recibe el nombre de cadena homogénea, en el cual las probabilidades son idénticas en cada paso.

Cuando observamos una cadena de Márkov finita con m posibles estados, podemos describirla como:

$$P_{ij} = P\,(X_n = j | X_{n-1} = i)$$

donde i,j = 1,2,...,m. Si Pij > 0 podemos asegurar que el estado i se puede comunicar con el j. La comunicación será mutua si cumple además Pij >0.

Para cada valor de i, obtenemos una distribución de probabilidad, P_{ij}, en la que podemos pasar a cualquier estado en cada paso, siendo excluyentes entre sí.

Los valores P_{ij} son las probabilidades de transición que satisfacen la condición $P_{ij} > 0$ para cada valor de i:

$$\sum_{j=1}^{m} p_{ij} = 1$$

Todos estos valores se pueden combinar originando la matriz de transición (T) de tamaño m × m:

Cada fila de la matriz será una distribución de probabilidad, dicho de otra manera:

$$T = [p_{ij}] = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1m} \\ p_{21} & p_{22} & \cdots & p_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m1} & p_{m2} & \cdots & p_{mm} \end{bmatrix}$$

Fig. 14: Matriz de transición. [23]

Una característica muy interesante es la probabilidad de llegar a E_j en n iteraciones, dada una distribución de probabilidad $\{p_i(0)\}$. Sabemos que $p_i(0)$ es la probabilidad de que inicialmente nos encontremos en el estado E_i, por lo que:

$$\sum_{i=1}^{m} p_i^{(0)} = 1$$

Será por lo tanto $p_j^{(1)}$ la probabilidad de que se alcance el estado E_j en una sola iteración, por lo que podremos llegar a la siguiente solución:

$$p_j^{(1)} = \sum_{i=1}^{m} p_i^{(0)} p_{ij}$$

Si lo expresamos de manera vectorial, agrupando todas las $_j$:

$$P^{(0)} = (p_1^{(0)}, \dots, p_m^{(0)})$$

Será a partir de ahora $P^{(0)}$ la distribución de probabilidad inicial y $P^{(1)}$ será con que probabilidad podremos alcanzar los estados $E_1, \dots, E_m$ en una iteración. Agrupando términos podemos concretar:

$$P^{(1)} = \left[p_j^{(1)} \right] = \sum_{i=1}^{m} p_i^{(0)} p_{ij} = P^{(0)} T$$

Y dicho de manera genérica:

$$P^{(n+r)} = P^{(r)} T^n$$

Esta teoría es ampliamente utilizada en multitud de campos desde la ingeniería genética hasta la música o la meteorología. En nuestro caso sentó las bases para nuestro método propuesto. En la Fig. 15: Ejemplo de CM . podemos ver como se podría estructurar una CM, esta estructura definirá el segundo pilar del SA cuando la tratamos conjuntamente con MC.

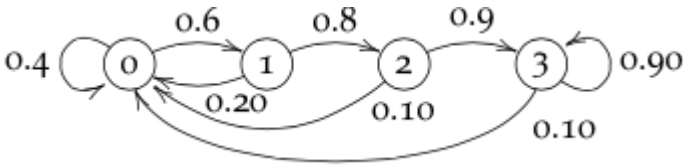


Fig. 15: Ejemplo de CM [13].

3.1.3 METRÓPOLIS-HASTINGS

Debe su nombre a su principal autor Nicholas Metrópolis, que publico un “paper” en 1953 introduciendo esta teoría. Además de W.K. Hastings que la extendió en 1970 hacia casos generalistas. Hay cierta controversia con la autoría del algoritmo, ya que el citado “paper” tuvo más autores y aún más que reclamaron su propiedad intelectual.

El MH es un algoritmo perteneciente a la rama MCMC. Es uno de los principales o más usados de estos métodos, lo que se debe a que la mayoría de los mismos pueden ser considerados como interpretaciones del MH.

El algoritmo está basado en la generación de una cadena de Márkov de un parámetro del que conozcamos la densidad en el siguiente estado, lo que llamaremos $f(\theta|x)$. Posteriormente necesitamos una densidad que sea fácil de muestrear, debe ser una densidad dependiente del estado anterior. Esta se pasará a llamar distribución propuesta o tentativa, y la denotaremos como $g(.|\theta)$.

El algoritmo se compone de los siguientes pasos:

1. Se debe establecer un valor inicial $\theta^{(0)}$, que normalmente se genera de manera aleatoria. Y no usaremos en el resto de iteraciones.
2. Se genera una muestra $\varphi \sim g(.|\theta^{(t)})$
3. Se calcula la probabilidad de aceptación de la muestra anterior como:

$$\propto \left(\theta^{(t)}, \varphi\right) = \min[\,1\,, \frac{f(\varphi|x).\, g\left(\theta^{(t)}\middle|\varphi\right)}{f\left(\theta^{(t)}\middle|x\right).\, g(\varphi|\theta^{(t)})}\,]$$

4. Tomar el siguiente valor de $\theta^{(t+1)}$ como φ, con una probabilidad de $\propto$. O dejar el valor actual en caso contrario, $\theta^{(t)}$.

Por tanto, podremos usar cualquier distribución, g, para implementar este algoritmo. Lo único necesario es que sea muestreable, pero cuanto más se asemeje a la distribución de las cadenas de Márkov, f, más rápido convergerán. En multitud de ocasiones se utiliza las primeras iteraciones del algoritmo para generar unas condiciones iniciales que, cuando se tienen suficientes, se borrar para no dejar el estado inicial (recordemos que se generó de manera aleatoria).

Para aproximar la forma de la distribución en su forma final, se puede recurrir a ecuaciones de kernel. El kernel hace que suavice los datos y su transición dentro de la cadena de Márkov.

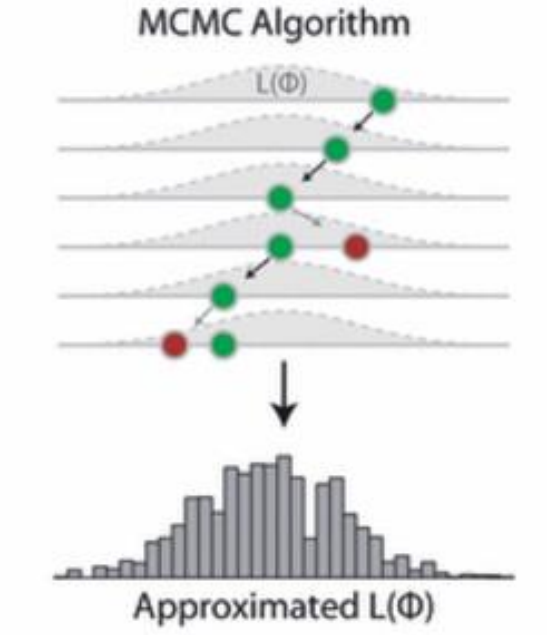


Fig. 16: Ejemplo de MH [14].

Las ventajas de este método son las mismas que el resto de métodos MCMC, sabiendo que el MH se puede aplicar a cualquier densidad objetivo. Los principales atributos que pueden favorecer o limitar la convergencia al óptimo vienen dados por:

- Debido a la correlación de las muestras, la función puede estancarse en un mínimo local y retrasar mucho la convergencia.
- Dada la correlación, pueden repetirse muestras. Por lo que sería ideal contar con una baja correlación en las aplicaciones del método.
- Debemos evitar las primeras iteraciones del algoritmo, ya que en estos primeros pasos no siguen la densidad objetivo.

Para evitar dichos problemas podemos usar versiones mejoradas del MH, tales como:

  - Múltiple Try Metrópolis (MTM). Es una modificación del MH que permite una convergencia más rápida. Mediante la modificación del parámetro de aceptación y el tamaño del salto de las transiciones.
  - MH con Delayed Rejection (MHDR). Se mejora el algoritmo original reduciendo el número de muestras rechazadas. En lugar de rechazar el candidato y quedarse

en la posición actual se propone crear un nuevo candidato con un nuevo parámetro de aceptación.

A continuación, en la **¡Error! No se encuentra el origen de la referencia.** vemos g ráficamente como el algoritmo se mueve por la distribución para tratar de aproximarla. Aceptando, con el color verde, los puntos que la cumples y rechazando los intentos no válidos, en rojo.

3.2 RECOCIDO

Todos los tipos de tratamientos térmicos del acero que calientan la pieza de trabajo y que además habitualmente tienen asociado un enfriamiento lento, se denominan recocido. Este enfriamiento se caracteriza por deberse aplicar de manera progresiva y uniforme. Normalmente para dicha labor se emplean hornos con sensores de temperatura para asegurar el control de dicho enfriamiento. Antes de disponer de dichos hornos, se asociaba una temperatura a cada color que adquiere el acero al calentarse, y se debía disponer la pieza en un lugar aislante del calor, usualmente el mismo horno.

Este tratamiento suele usarse como tratamiento inicial para ablandar el acero. Su finalidad suele ser la de suprimir los defectos del temple[5] y mejorar las propiedades mecánicas iniciales del material, tales como dureza, resistencia, tenacidad, resiliencia, ductilidad, etc. También logra eliminar la acritud, afinar el grano y homogeneizar la estructura.

Es necesario entender este concepto para saber cómo haremos posteriormente nuestro algoritmo, por esa razón debemos estudiar las etapas del recocido, la recuperación

[5] El temple es un tratamiento térmico que se caracteriza por enfriamientos rápidos en un medio, sea agua, aceite o aire, para transformar la austenita en martensita. Aumenta la dureza y la resistencia mecánica, disminuye la tenacidad, reduce la plasticidad y modifica propiedades eléctricas, magnéticas y químicas.

de tensiones internas, como cristaliza el acero, y la influencia de la temperatura y el tiempo en dicho proceso.

3.2.1 ETAPAS DEL RECOCIDO.

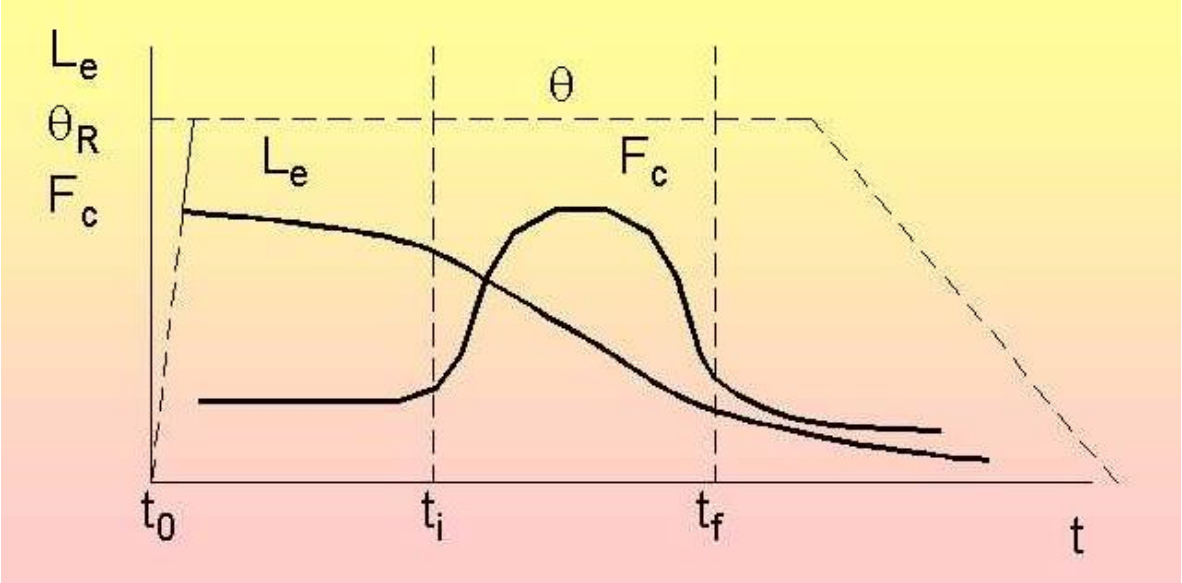


Fig. 17: Gráfica de las etapas del recocido. [15]

Durante el proceso de recocido isotermo a 450ºC se pueden observar tres etapas cuando se compara la función del flujo térmico (Fc en la Fig. 17), lo que nos reduce el proceso en las siguientes etapas:

1. Primera etapa. Comprendida entre el instante inicial y el tiempo en el que la Fc se mantiene constante. Esta etapa se denomina restauración o recocido contra tensiones.
2. Segunda etapa. Comprendida entre el tiempo en el que aumenta la función de Fc y el momento en el que vuelve a un valor próximo a cero. Re-acoplamiento de los cristales a estructura de mínima energía libre. Se conoce como recristalización.

3. Tercera etapa. Se extiende durante los tiempos superiores al final de la anterior etapa. Disminuye la energía libre y la dureza debido al aumento del tamaño del grano (Obsérvese en la Fig. 18).

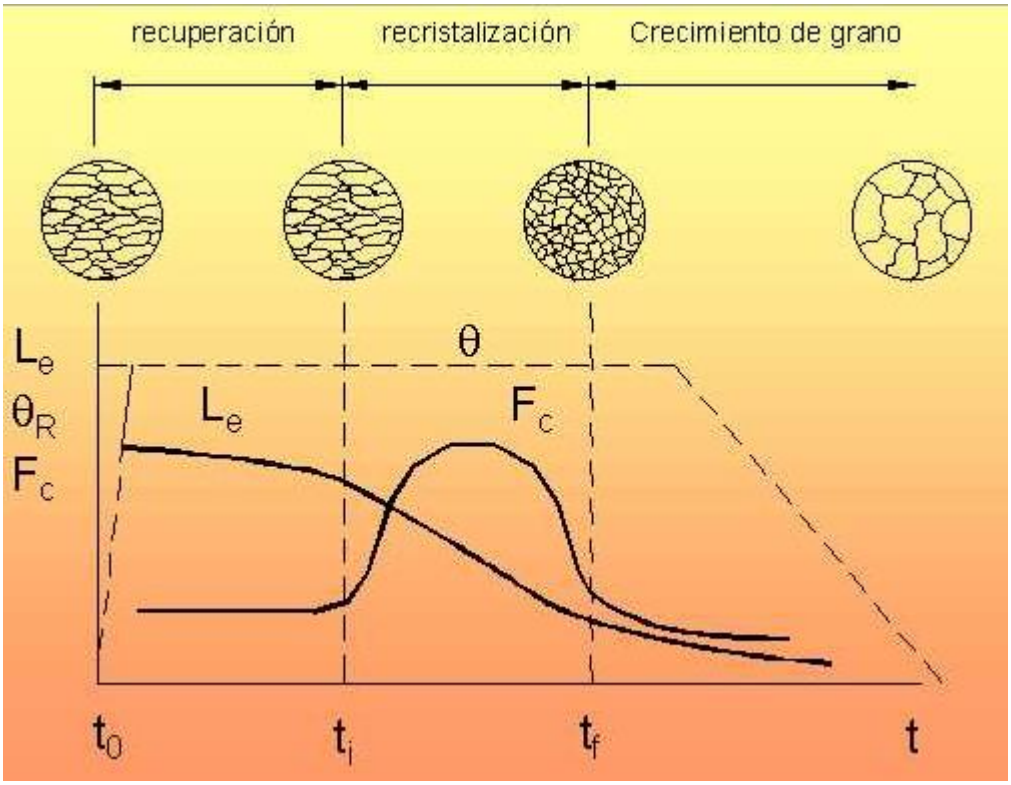


Fig. 18: Evolución de la microestructura a lo largo del tratamiento de recristalización. [16]

3.2.2 CARACTERÍSTICAS DE LA RECRISTALIZACIÓN DEL ACERO.

Si observamos la microestructura cristalina de la última etapa, podremos observar como los granos empiezan a tomar formas geométricas de lados iguales.

La cristalización nueva viene dada por la disminución del límite elástico en la pieza ya endurecida. Esta libera energía con la siguiente ecuación:

$$E_p = \int_{t\infty}^{t} F_c . dt$$

Con un valor máximo en el que la función F_c se reduce a un valor nulo. Por lo tanto, dicha energía liberada es un indicador de la masa que ha recristalizado. Podemos concluir que el recocido disminuye el tamaño del grano y se puede producir ciertas microestructuras controlando la velocidad de enfriamiento del metal, lo que se traduce en la minimización de la energía interna que posteriormente se libera.

3.2.3 INFLUENCIA DEL TIEMPO Y LA TEMPERATURA EN LA RECRISTALIZACIÓN

Se va a procurar establecer una relación entre el tiempo de cristalización y la temperatura del recocido. Si vemos cómo evoluciona el tiempo que tarda en recristalizar un acero totalmente, en nuestro ejemplo el F1110. Observamos que este tiempo (denominado t_{rf}) es la suma del tiempo que tarda en aliviar las tensiones (t_{rs}) y el empleado en generar nuevos mono-cristales (t_r). Como se puede observar en la siguiente ecuación:

$$t_{rf} = t_{rs} + t_r * (0.95)$$

Si observamos la Fig. 19,que nos muestra la relación entre las temperaturas de recocido y los tiempos de alivios de tensiones, podremos afirmar una evolución paralela de las curvas dadas por la temperatura ante el mismo tiempo empleado por el acero en recristalizar totalmente. O bien, el aumento de la temperatura de recristalización consigue disminuir el tiempo necesario para recristalizar totalmente el acero. Tiempos y temperaturas mantienen una correlación exponencial inversa, siendo altamente sensible a la variación de temperatura, o como consecuencia la energía.

Esto podemos explicarlo si sabemos que deberá disminuir menos la temperatura para poder cristalizar totalmente el acero. Lo que aplicaremos posteriormente como un límite para nuestra optimización de la función de coste.

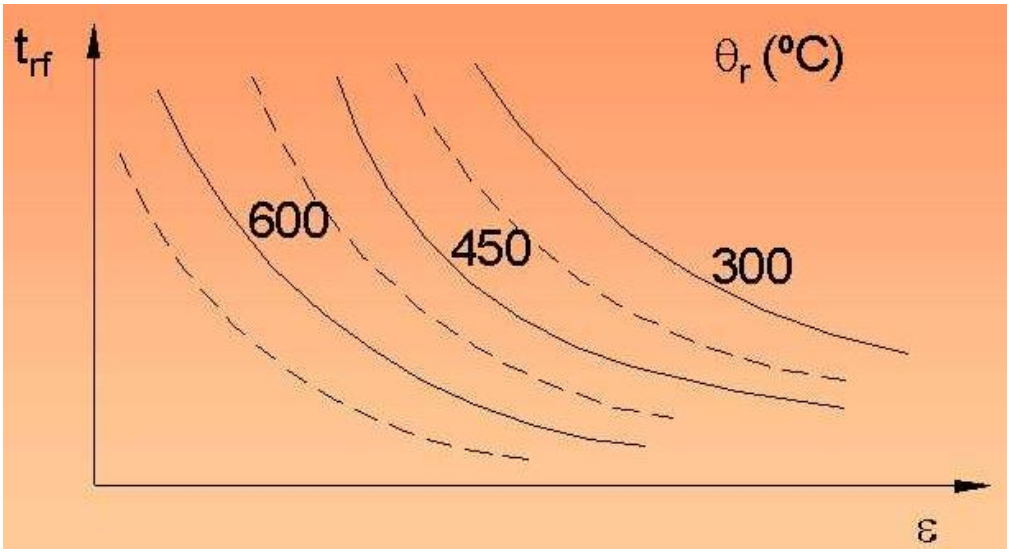


Fig. 19: Correlación de la deformación correspondiente a cada temperatura y el tiempo empleado en recristalizar. [17]

Podemos por tanto concluir que el recocido es un tratamiento térmico que busca alterar las propiedades del metal mediante un proceso que consiste en tres pasos:

- Calentar el metal hasta una determinada temperatura.
- Mantener dicha temperatura en función de las propiedades buscadas.
- Dejar enfriar lentamente, disminuyendo la temperatura controlada y progresivamente.

El proceso finaliza cuando el metal haya cristalizado totalmente o bien alcance la temperatura ambiente. Esto nos servirá como fundamento teórico para poder entender como hemos fundamentado el SA. A modo de introducción buscamos una función que minimice la temperatura, determinando cuando es mejor bajarla y cuando no de modo que minimice la energía empleada.

Como ya hemos explicado anteriormente y a modo de resumen, el fenómeno se inspira en el recocido del acero, esta técnica consiste en calentar el material en estado sólido y luego enfriarlo lentamente con el fin de conseguir propiedades físicas diferentes. La explicación física a este fenómeno viene dada por los átomos que aumentan su energía y comienzan a ser capaces de cambiar su posición inicial, lo que llamaríamos mínimo local de energía. Al enfriarlo lentamente somos capaces de obtener una mayor probabilidad de recristalizar las configuraciones de dichos átomos con menor energía que la inicial, lo que supondría el mínimo global.

3.3 SIMULATED ANNEALING. PRINCIPIOS TEÓRICOS.

Como hemos introducido anteriormente, el SA es un método de la rama MCMC. Este algoritmo de búsqueda trata de encontrar un mínimo global, pero no necesita de condiciones iniciales, el objetivo del mismo es precisamente llegar al valor óptimo desde un estado inicial aleatorio.

El algoritmo trata de minimizar la temperatura usando la energía mínima posible. Existes métodos alternativos para este objetivo, si usáramos métodos deterministas podríamos atascarlos en un mínimo local fácilmente. Si lo hiciéramos de manera completamente aleatoria podríamos llegar al valor óptimo de manera ineficiente. El SA combina ambos estilos y propone tanto un seguimiento de la trayectoria como un camino aleatorio para aunar tanto eficiencia como completitud.

El SA evalúa, debido a su herencia heurística, los vecinos del estado actual. Asume con cierta probabilidad un cambio hacia un nuevo estado o mantenerse en el mismo. Como hemos introducido en el recocido [0]3.2 Recocido, el espacio de estados son los átomos del material en el instante en el que lo evaluamos. Si un átomo se desplaza tomaría el título de

vecino en este caso, originando una nueva comparación de estados de energía. Este fenómeno se repetirá hasta alcanzar un estado de mínima energía en el cuál todos los átomos hayan alcanzado su cristalización y no sea posible encontrar un estado de energía menor.

Por consiguiente, ahora podemos definir vecindario como todo el espacio de estados que puede adoptar un átomo en su próximo movimiento. Estos se crean mediante métodos de MC [0]. El SA tiene su principal ventaja a la hora de comparar dichos vecinos. Con un método cualquiera heurístico podríamos movernos hasta un mínimo local y no poder evaluar un vecino mejor, con el SA para encontrar una solución óptima global del problema, puede aceptar una solución peor que evite el estancamiento en una solución local con cierta probabilidad. Esto es fundamental, ya que si nos centramos rápidamente en el mejor candidato podremos no estar abarcando todo el espacio de exploración necesario para resolver óptimamente el problema.

Entonces, el cómo evaluamos esas transiciones hacia un estado mejor o peor es la pieza fundamental de este método. Esta probabilidad va a depender de la diferencia de energía entre ambos estados (ΔE) y de la temperatura a la que nos encontremos T. Estas variables son las que nos permiten controlar el recocido y análogamente su simulación. Si el ΔE es negativo la nueva transición bajará la energía, lo que nos beneficia en nuestra búsqueda del mínimo global y por tanto debemos aceptarla siempre. Si el ΔE es positivo implicará que el siguiente estado aumentará la energía y nos alejará del mínimo global y no debemos aceptarlo. Aun así, no debemos establecer una posibilidad de transición de cero ya que correríamos ese riesgo que ya hemos comentado, el de bloquearnos en una solución local, y debeos aceptar una peor solución con el criterio que expondremos más adelante.

La probabilidad de transición debe cambiar a lo largo de la búsqueda del mínimo. De tal modo, cuando nos aproximemos a una temperatura baja debemos disminuir tal probabilidad hasta cero de manera asintótica. Si la temperatura llegara al mínimo, cero en nuestro caso, debemos no aceptar ningún cambio desfavorable y limitarnos sólo a buscar estados de menor energía. La temperatura por tanto es la principal característica en el

control del algoritmo, en estados de alta temperatura aceptaremos un cambio fácilmente. En estados de baja temperatura aceptaremos un cambio de manera poco habitual.

Para buscar una nueva temperatura, hemos de usar una función que guarde la mejor solución y proponga un vecino de la misma. Suele utilizarse una función exponencial donde se varíe la temperatura en relación a un parámetro de recocido (k), como se indica en la siguiente ecuación:

$$T = T_o . 0.95^k$$

Siguiendo este criterio podemos asegurar que en cada iteración del algoritmo la temperatura va decreciendo una tendencia del parámetro de recocido. Por su constitución, el último valor posible de T que puede ofrecernos este criterio es cero. Con este sistema, en los primeros pasos el algoritmo se moverá libremente por el espacio de búsqueda a explorar y se irá concentrando, en cada iteración, en estados de menor energía pudiendo ignorar pequeños aumentos de temperatura. Para finalmente cambiar a estados mejores y alcanzar el mínimo global. La probabilidad de alcanzar la solución óptima global puede ser del 100% si se corre el algoritmo el tiempo necesario, pero esta no suele ser lo habitual debido a los grandes tiempos de carga no deseados en el ámbito de los algoritmos.

Como hemos venido haciendo en el resto del trabajo, vamos a intentar deducir una serie de pasos para ejecutar este algoritmo. Primero, describiremos en pasos como funciona [18] para posteriormente centrarnos en un seudocódigo que utilizaremos para resolver nuestro problema particular.

El SA suele conllevar los siguientes pasos:

1. Generamos un estado inicial aleatorio.
2. Seleccionar si el nuevo estado es mejor o peor que el actual. Si es mejor se convierte en el actual. De lo contrario, si es peor, puede convertirse en el actual si cumple la función de aceptación, esta probabilidad (q) viene dada por:

$$q = \min\{1 \; y \; e^{\frac{-\Delta E}{T}}\}$$

Si ΔE es positivo y T (que por las condiciones del algoritmo no puede bajar de cero) también lo es, esta probabilidad debe rondar entre 0 y 0.5. Cuando aumentamos ΔE también supone un aumento de esta probabilidad, esto favorece la aceptación de peores puntos en los primeros pasos del algoritmo cuando la temperatura es mayor.

3. Bajamos la temperatura, guardando el mejor punto que hayamos encontrado. El método para disminuir dicha temperatura puede ser:
    i. Exponencial: $T = T_o . 0.95^k$
    ii. Proporcional: $T = T_o / k$
    iii. Logarítmica: $T = T_o / \log(k)$, o bien: $T(t) = k / \ln(1 + t)$

4. Reajustamos el parámetro de recocido (k), este parámetro depende de los gradientes que hayamos definido en la función objetivo. Sigue la siguiente ecuación:

$$k_i = log\left(\frac{T_o}{T_i} . \frac{max_j(S_j)}{S_i}\right)$$

Dónde k_i es el parámetro de recocido para la componente i.

T_o es la temperatura inicial de la componente i.

T_i es la temperatura actual de la componente i.

S_i es la pendiente del objetivo en la dirección i.

O bien k puede ser fijada de forma experimental.

5. El algoritmo retornará al punto 2 hasta que cumpla una de las condiciones de parada. Estos criterios suelen ser:
    i. Llega a la mínima tolerancia. Entendiendo tolerancia como la distancia máxima que aceptamos hasta la condición óptima. Por ejemplo, este valor suele rondar el orden de 10^{-6}.
    ii. Que hayamos alcanzado las máximas iteraciones que estimemos como límite.

iii. Que hayamos sobrepasado las máximas evaluaciones de la función, también es un límite que deberemos establecer empíricamente.
iv. Que lleguemos al tiempo máximo. Este límite que debemos poner bajo nuestro criterio, es muy útil para evitar cuelgues de ejecución si se demora demasiado.
v. Por último, cuando hayamos alcanzado el objetivo óptimo, una vez lleguemos al mínimo global podemos parar de iterar ya que no encontraremos un punto mejor.

Si estos pasos se hacen adecuadamente, podremos llegar sin error a un estado en el que el metal obtenga la mínima energía. Se puede demostrar que si el algoritmo disminuye T lo suficientemente despacio alcanza el óptimo global.

```
Función SA (problema), devuelva (solución óptima)
{
Actual= estado_inicial (problema); <- Puede ser aleatorio.
T= temperatura_inicial (problema); <- Debe ser lo suficientemente alta para que lo consiga
                                     resolver.
Bucle exterior, repetir hasta T=0
        {
        Estado=actual;
        Bucle interior i = 0 : tope
                {
                Intento= sucesor_aleatorio (estado);
                Si  ΔE < 0 entonces
                        {
                                Estado=intento;
                        } sino
                        {
                                q = min {1 y e^(-ΔE/T)}    ;
                                Si aleatorio (0,1) <q entonces
                                        {
                                        Estado=intento;
                                        }
                        }
                i = i + 1;
                }
        Actual=estado;
        T =  T_o. 0.95^k; <- Puede ser cualquier función de actualización de T.
        } devuelve actual;
}
```

Fig. 20: seudocódigo del SA.

Por tanto, el algoritmo podemos deducir que consta de dos bucles:

- Un bucle interno, en el que iremos actualizando las nuevas configuraciones de la temperatura, a partir de la anterior. Esta reproduce un camino aleatorio para una temperatura dada, y si deberemos interrumpirlo si hemos alcanzado la condición de equilibrio o el máximo de pasos.

- Un bucle externo en el que disminuyamos la temperatura, garantizando que nos acercamos a la solución óptima. En él iremos actualizando la variable actual, lo que constituirá una sucesión de MH.

Teniendo en cuenta toda esta información, se propone un seudocódigo (consultar Fig. 20) en el que podremos basar para la programación, en cualquier lenguaje, de este algoritmo [19]. Esto sólo es una guía general, para cada problema podrán existir dificultades añadidas que necesiten de ajustes extra, como sucede en nuestro problema.

Podemos observar en el código de la Fig. 20 las siguientes peculiaridades:

- El objetivo es encontrar la mejor configuración de estados para una temperatura dada.
- Explora los vecinos hasta el tope que hayamos fijado.
- No discrimina los peores sucesores ni elige los mejores, elige uno al azar por lo que debe recorrer todo el espacio de posibles soluciones.
- Un intento puede ser elegido según su energía y de la temperatura en la que nos encontremos:

  Si T es muy alta todos los intentos tienen la misma probabilidad de ser elegidos. Mientras que si la T es muy baja sólo las de mínimo coste serán candidatas (distribución de Gibs).

Como puede observarse en las Fig.21 -23, vamos[6] a ofrecer un ejemplo visual de cómo va evolucionando el algoritmo con el tiempo, a modo resumen. En los primeros pasos (1-7) podemos observar en la Fig. 21 como empieza generando una posición inicial (azul) aleatoria, luego genera un vecino aleatorio (verde) y los evalúa. Es menor el coste del

[6] Ha sido confeccionado por mí, basándome en un gift que ha desarrollado Stuart Reid [35].

segundo y decide moverse, como era de esperar. Opera de esta manera en dos ocasiones más.

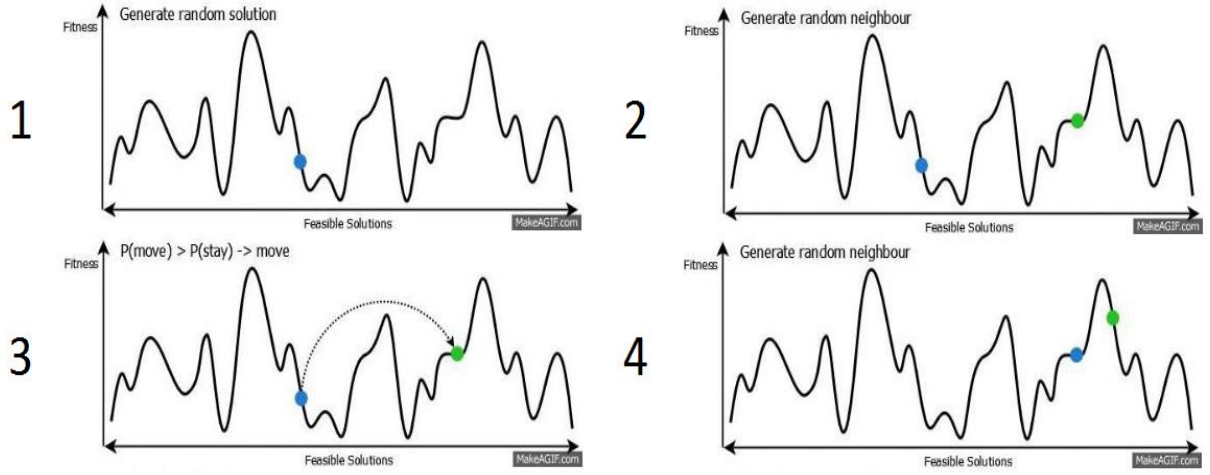


Fig. 21: Primeros 4 pasos del ejemplo visual de SA.

Como podemos observar en la Fig. 22, en el último caso (7-8) a pesar de ser peor valor, se cambia de posición, pero deja almacenada la mejor (en color rojo).

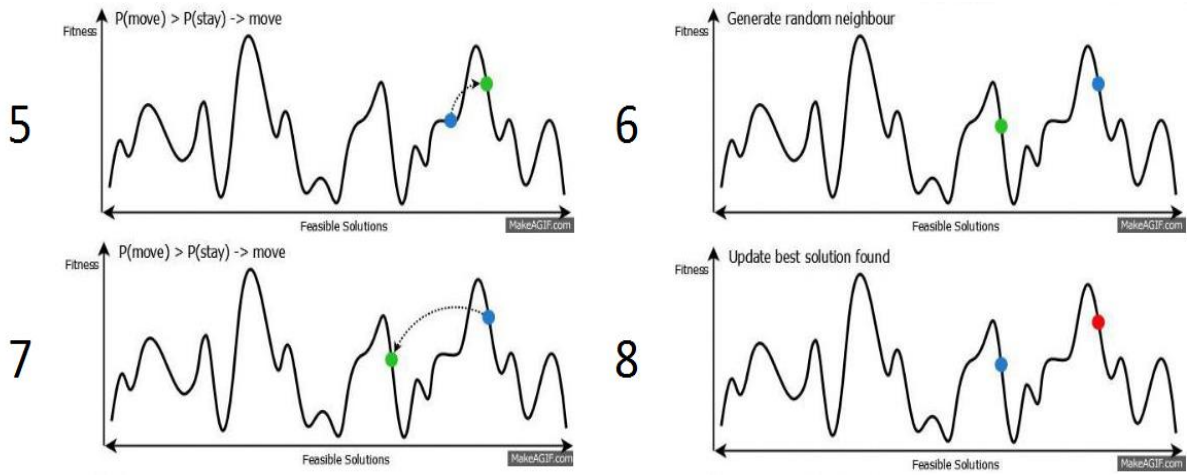


Fig. 22: Segundos 4 pasos del ejemplo visual de SA.

Por último, en la Fig. 23 vemos como vuelve a ejecutar las anteriores condiciones de la misma manera, evaluando y moviéndose por el espacio de estados, saltando entre soluciones locales para descubrir cuál conlleva la solución global.

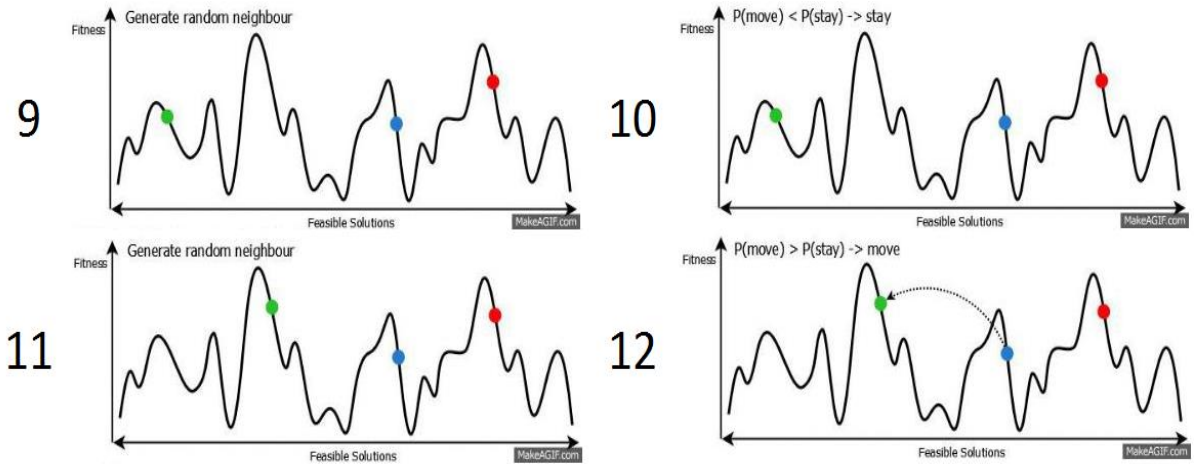


Fig. 23: Terceros 4 pasos del ejemplo visual de SA.

Finalmente, en la Fig. 24 vemos como al encontrar la mejor solución, vuelve a intentar cambiar, pero como mejora el mejor punto histórico lo almacena y concluye, por cumplimiento de la tolerancia, el algoritmo.

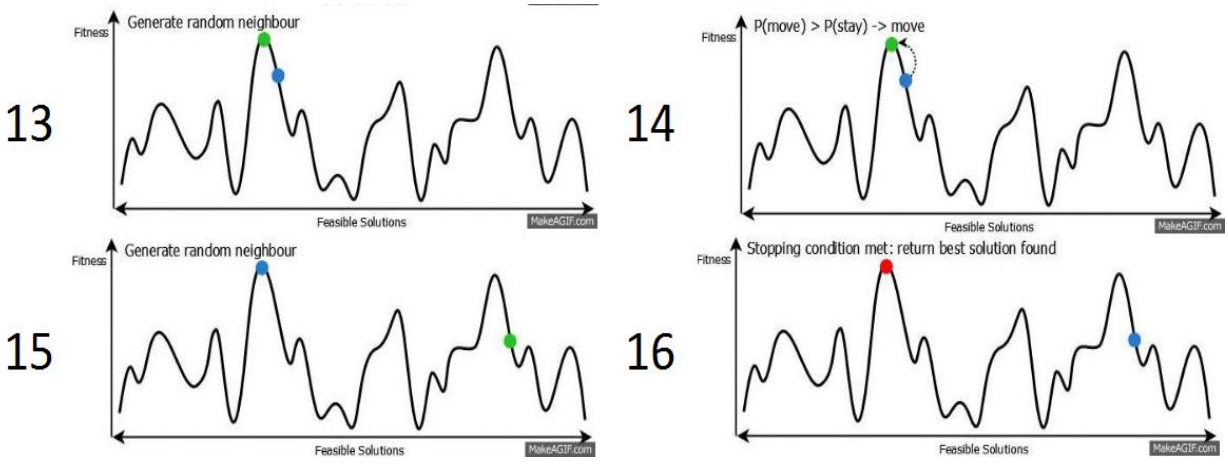


Fig. 24: Últimos 4 pasos del ejemplo visual de SA.

Tras esta última explicación, creo que seremos capaces de resolver nuestro problema particular. A lo largo del mismo nos encontraremos problemas menores para los que necesitaremos readaptar nuestro algoritmo teórico. Usaremos el código básico que propusimos anteriormente, en la página 42. En simbiosis con los pasos de la página 40.

Podemos consultar los pasos realizados por nuestro código en el anexo C.

4. LOCALIZACIÓN GLOBAL PARA ROBOTS MÓVILES

A continuación, vamos a presentar nuestro problema particular. En él se nos plantea resolver la localización de un robot móvil en cualquier escenario.

Un robot autónomo precisa de saber su localización para ejecutar múltiples tareas como la navegación o manipulación de objetos. Por lo tanto, uno de los problemas más importantes de un robot autónomo es su LG, que consiste en buscar sus coordenadas en un entorno conocido sin información de su localización inicial. Es posible distinguir entre dos diferentes sistemas de LG según su fuente de información.

Por un lado, los sistemas de posicionamiento que reciben las señales de una fuente externa. Un ejemplo es el sistema de posición global (comúnmente conocido como GPS).

Y por otro lado los sistemas de auto-posicionamiento que se basan en sensores implementados dentro del robot. Algunos ejemplos típicos son módulos de localización basados en rangos láser con los que obtienen escaneos en dos (2D) y tres dimensiones (3D).

Este problema debe incluirse en la segunda opción (3D) ya que nuestro robot trabaja en sitios cerrados y la información viene dada por un sensor de rango láser.

En nuestro trabajo previo, se desarrolló un algoritmo basado en el método de DE, que resuelve el problema de LG en entornos 2D y 3D. Estos métodos son técnicas de optimización basadas en la representación de la posición y orientación del robot mediante un conjunto de estimaciones de posibles localizaciones (población) ponderadas por una función de coste. El conjunto de soluciones evoluciona en el tiempo para integrar la información del sensor y la información del movimiento del robot.

Se nos plantea el mismo problema para tratar de alcanzar una solución similar, o mejor si fuera posible con otro método, el SA. Esta técnica de optimización se basa en la

representación de la pose del robot (posición y orientación) mediante un conjunto de puntos basado en los métodos MCMC que van cambiando con cada iteración, no evolucionan, acercándose a las posibles localizaciones óptimas que finalmente serán descartadas cuando se encuentre un óptimo global. El criterio que moverá la aceptación de mejores o peores soluciones debe ser el resultante de la función de costes que elijamos, de las facilitadas por el tutor. Esto lo veremos con detalle más adelante.

Las ventajas más importantes de nuestro método han sido explicadas en los apartados anteriores, citados arriba. El algoritmo puede adaptar sistemas dinámicos no lineales arbitrarios, características de sensores y ruido no Gaussiano. Debido a que el conjunto de soluciones no trata de aproximar la distribución de densidad posterior, no requiere de ninguna premisa de la forma de la densidad posterior, como hacen las aproximaciones paramétricas. El SA concentra los recursos del hardware en las áreas más importantes, asignando el conjunto de soluciones a las áreas más interesantes, según la función de coste.

4.1 MAPA

El SA, junto con la función de costes y el resto del programa ya realizado, emplea un mapa con cuadrícula para establecer la posición actual del robot, la real. En el mismo mapa debemos estimar su localización mediante el SA. Para poder trabajar, el tutor ha facilitado 5 tipos de mapas con los que iremos probando el código, que os presento a continuación:

1. Mapa total:
   Perteneciente a un pasillo completo del edificio número 1 de la Universidad Carlos III de Madrid en el campus de Leganés.

Fig. 25: Mapa total. Todas las unidades en celdas.

2. Mapa Parcial:

   En este mapa parece que podemos usar el mapa total, pero si nos fijamos con detenimiento sólo podemos utilizar hasta la unidad 500 del eje x, lo que corresponde a la mitad. Perteneciente a medio pasillo completo del edificio nombrado anteriormente.

Fig. 26: Mapa parcial. Todas las celdas en unidades.

3. Mapa para realizar test:

   Perteneciente a un conjunto de despachos del mismo edificio que hemos citado en los anteriores puntos.

Fig. 27: Mapa de test. Todas las unidades en celdas.

4. Mapa de gran tamaño:

   Perteneciente a toda la planta 3 del edificio citado anteriormente.

Fig. 28: Mapa de planta completa. Todas las unidades en celdas.

5. Mapa real:

   Perteneciente a un barrido parcial de un segmento de dicho pasillo, citado anteriormente, con más detalle.

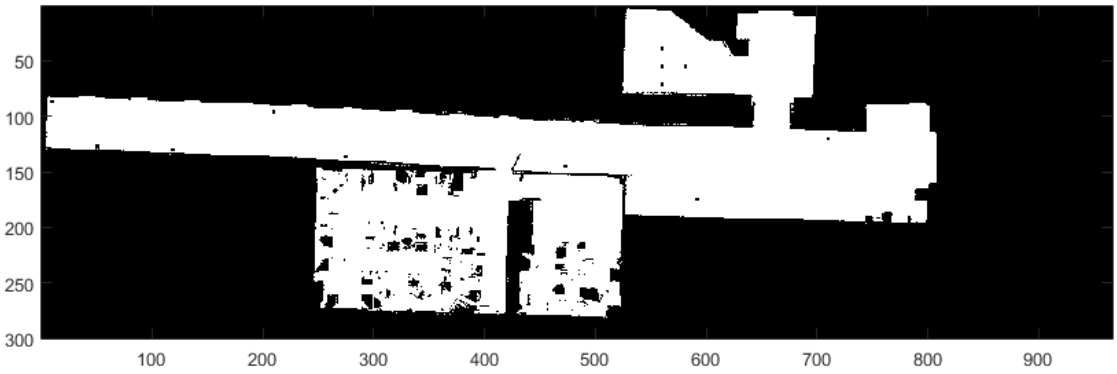


Fig. 29: Mapa de fragmento de pasillo. Mapa real. Todas las unidades en celdas.

El primer problema que nos encontramos, como ya podréis deducir, es qué mapa emplear y que características posee. En este trabajo, vamos a representar los resultados en el primer mapa, Fig. 25, por ser suficientemente significativo y poder comparar los resultados de todas las pruebas en el mismo escenario. Debo aclarar que se han probado todos, y el algoritmo no diferencia entre unos y otros, por lo que resulta igual de preciso realizarlo en unos u otros.

Con respecto a las características que posee, podemos destacar que hay muchas habitaciones, o despachos, totalmente idénticos entre sí. Puesto que nuestro robot percibe el espacio mediante un sensor laser, no puede distinguir sin más información sensorial si se encuentra en uno u otro en algunas ocasiones. Sabrá su posición real pero no la podrá estimar al 100% en algún caso. Para este problema no tendremos más soluciones teóricas, pero en la realidad no lo supone. Puesto que al igual que las personas, en el primer barrido no será capaz de saber en qué despacho se encuentra con certeza, pero con el primer movimiento si debería ser capaz de completar la información y volver a ubicarse. También posee dos áreas idénticas a ambos lados que corresponden a los baños, y otras tres idénticas que se identifican claramente, las escaleras. En estos puntos nos costará

probablemente más de un intento para poder ubicarlo. Pero se encuentra dentro de los resultados esperados.

4.2 FUNCIÓN DE COSTE

Como hemos indicado con anterioridad, el SA debe moverse según la función de coste. Esta debe calcular el error, o el coste, que existe en cada aproximación que queramos realizar. No hay un único método para calcularla, y algunas son más rápidas mientras que otras son más precisas. Para su uso con el SA, podemos utilizar la que queramos, ya que nosotros usaremos el error que nos devuelvan, y a pesar de que puede influir en el tiempo empleado en hallar la solución global, no resulta determinante y podemos obviar esta última consecuencia para logar nuestro objetivo, resolver el problema de LG.

El SA procesa una función de coste que se apoya en variaciones del error entre las observaciones y las medidas estimadas. Estos métodos usan la información de la posición vertical de los elementos del entorno (paredes, columnas, puertas, muebles…) para obtener la función de coste que permite distinguir entre las diferentes localizaciones. La función de coste más utilizada para la localización de robots móviles es la cuadrática. Sin embargo, hay algunos autores que han considerado diferentes funciones. En el trabajo previo han aplicado la distancia Manhattan (Norma L1) concluyendo que es una aproximación más factible en entornos con objetos dinámicos, personas, etc… Otra alternativa publicada recientemente [19], es la divergencia KL que puede ser definida como “una medida no simétrica de la diferencia entre dos distribuciones de probabilidad P y Q”. Propuesta por Kullback y Leibler en 1951. Entre otras de una lista que daremos a continuación.

Se ha asumido que el entorno se representa con un mapa con cuadrícula para saber la ocupación del entorno, que es el más típico para aproximaciones LG. En este tipo de mapas, el entorno 2D se discretiza usando una cuadrícula de celdas del mismo tamaño donde cada celda tiene asociado un valor entre cero y uno, que representa la probabilidad de estar ocupado.

El principal problema de estos métodos es distinguir una oclusión, esta sucede cuando un obstáculo sin modelizar origina un error cuando se compara la posición real del robot y la estimación óptima en el mapa. Cuando la medida real es ligeramente inferior o similar a la estimada puede ser debido a una oclusión, pero también puede causar esta diferencia el ruido de la medición. Esto es importante tenerlo en cuenta a la hora de comparar los resultados si usamos diferentes funciones de coste.

En este caso, el trabajo previo se compone de varios tipos de dicha función, que se presentan a continuación:

1. L2 Norm. Sum of the squared errors (Default).
2. L1 Norm. Sum of the absolute values of the error (Mahalanobis).
3. Kullback-Leibler Divergence based.
4. Density Power Divergence based.
5. Hellinger Distance based.
6. L2 Norm from Probability Distributions
7. L (Variable Exponent) Norm from Probability Distributions.
8. Generalized Kullback-Leibler Divergence based.
9. Itakura-Saito Divergence based.
10. Jensen-Shannon Divergence.

El método que utilicemos para calcular el error es indiferente para resolver nuestro problema puesto que todos han sido implementados y corroborados, por lo que nos darán un error que podremos usar para movernos en nuestro espacio de soluciones.

Lo que debemos saber es qué inputs necesitan estas funciones para poder usarlas en nuestro código. Estos son:

1. Vector con los elementos que contienen las distancias medias por el láser de la verdadera localización.
2. Vector con los elementos que contienen las distancias del láser estimadas por nuestra función, SA, que son candidatas a la solución óptima.
3. Versión a utilizar de la función de coste, listado anteriormente. Podemos incluir cualquiera de los 10 métodos propuestos.
4. Número de medidas horizontales en cada medición del láser.

Debemos agregar estos inputs cuando realicemos la llamada desde nuestro algoritmo a esta función, de lo contrario obtendremos errores en el código y no podrá ser ejecutado.

Por otro lado, el único output que nos devuelve esta función es el error, o coste del candidato analizado.

Esto implica que en cada punto que queramos evaluar como candidato deberemos asociar un error mediante esta función, y así poder comparar si es mejor o peor que el actual.

Podremos consultar el código de dichas funciones en el anexo D.

Para comparar los resultados obtenidos vamos a usar la norma L2, o suma de los errores cuadráticos, que dada su sencillez nos ayudará a generar el error de manera más inmediata. Si buscáramos una versión más desarrollada del código quizás deberíamos realizar los experimentos con la divergencia Kullback-Leibler, o similar. Puesto que no es representativa la diferencia para nuestro objetivo, optaremos por la opción citada anteriormente.

5. SOLUCIÓN PROPUESTA

La solución que se expondrá a continuación es una variante del SA aplicada a nuestro problema particular de LG. Se compondrá de los fundamentos teóricos de los métodos de búsqueda heurísticos, basados en la optimización de principios MCMC y en la aplicación del algoritmo de SA.

Partimos de la base del código propuesto por nuestro tutor, empleado para algoritmos de búsqueda evolutiva. Esto nos marcará muchas variables de entrada y salida que deberemos adaptar, aunque en algún caso no usemos. Pero nuestro código debe ser compatible con el resto de algoritmos de búsqueda para poder incluirlos en el programa y que, en cualquier momento, podamos comparar su ejecución, optar por uno u otro, o poder cambiarlos y definir qué función de coste es la mejor en todas las situaciones. Por ello y sin ánimo de entorpecer el trabajo previo, nos adaptaremos a él.

La solución debe usar los siguientes inputs:

- NP, hace referencia al número de población. En nuestro caso no pretendemos usar una población puesto que no es un método evolutivo. En nuestro caso debemos tantear el espacio de posibles soluciones según el método de MC. Este método implica generar un conjunto de puntos candidatos a ser solución que posteriormente trataremos como cadena de Márkov. El cuál almacenaremos en dicha variable a pesar de no ser una población. De la misma manera que realizamos nuestro experimento, este parámetro puede ser determinante en nuestro resultado y puede ser definido de una manera más precisa en futuras actualizaciones (consultar experimento en secciones 0, 0, 0).
- F, hace referencia al factor de variación diferencial. En nuestro caso no es utilizado.
- CR, hace referencia a la constante de cruce. En nuestro caso no es utilizado.
- T, hace referencia a la constante de translación. En nuestro caso no es utilizado.
- D, hace referencia al número de cromosomas. En nuestro caso no es utilizado.

Los siguientes inputs son facilitados por la selección manual del usuario del código directamente, o debido a sus elecciones en el código principal.

- Err_dis, ruido del sensor. Este parámetro nos lo ajustará el usuario final del código manualmente. Posteriormente, en la adaptación al robot real, será dado por el mismo automáticamente y no será necesario su manipulación.
- Map, mapa escogido por el usuario donde elaboraremos nuestro algoritmo, y visualizaremos posteriormente.
- Mapmax, característica física del mapa escogido. Máximas dimensiones en celdas, que usaremos como límites de nuestras nuevas soluciones candidatas. Para evitar salirnos del mapa.
- Mapmin, característica equivalente a mapmax con las mínimas dimensiones del mapa.

- Versión_de, hace referencia a la versión empleada para buscar el máximo. Como hasta ahora solo había búsquedas evolutivas poseía ese nombre. Podremos cambiarlo en el futuro si estuvieran de acuerdo el resto de usuarios del código. Para evitar conflictos lo hemos dejado como está.
- Versión_fitness, hace referencia a la versión empleada para calcular el coste.
- Iter_max, hace referencia al límite de iteraciones del código.

Los siguientes inputs serán facilitados por la función principal:

- Sensor_range, hace referencia al rango del sensor láser.
- Sensor_res, hace referencia a la resolución angular del sensor en radianes.
- Num_measurements, hace referencia al número de medidas horizontales en cada medición del láser.

Como hemos previsto, hay algunas entradas que no nos son necesarias, pero debemos no eliminarlas para asegurar el funcionamiento del resto del programa.

Como es un método heurístico, con herencia de los MCMCs tan sólo usaremos el estado actual y generaremos uno nuevo para comparar si tiene menor coste. Por ese motivo no necesitamos dichos inputs.

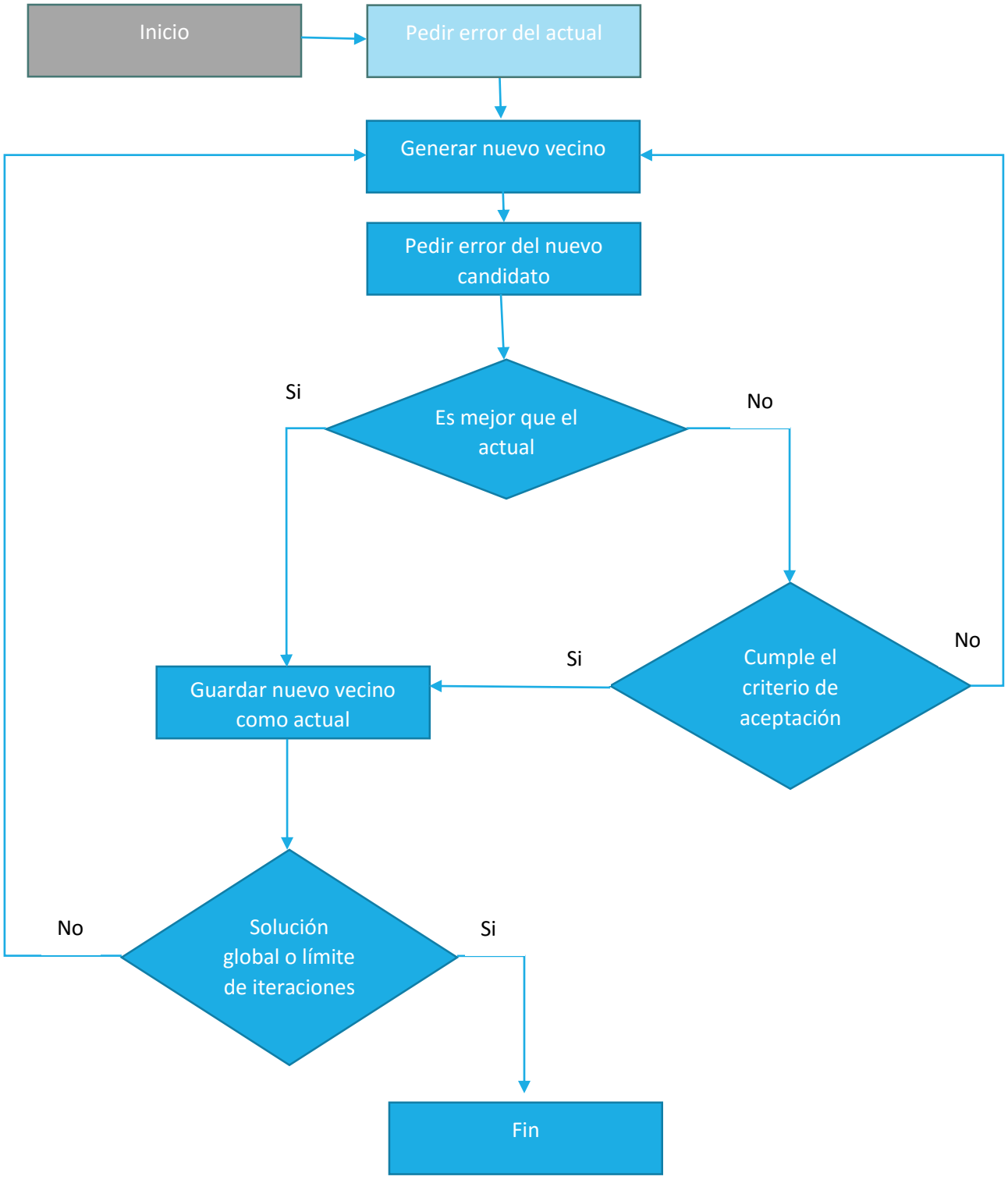
En cuanto a las salidas, nuestra función debe generar los outputs siguientes:

- Bestmen, hace referencia al vector con las soluciones de la LG, dónde el primer elemento debe ser el coste que lleva asociada dicha solución.
- Error, hace referencia al error correspondiente a la mejor solución, o la solución óptima del problema, que será impreso por pantalla como solución final.
- Population, conjunto de soluciones después de encontrar el óptimo global o superar algún límite de iteraciones.
- F, idéntico al input. No se usa en nuestro caso.
- NP, idéntico al input. En una de las mejoras realizadas en el código, cuando nos acercamos a la solución óptima reducimos el espacio de soluciones a recorrer

para favorecer un movimiento hacia la mejor solución y debemos sacar, actualizando, el número de posibles soluciones de la función.

Dados toda esta información debemos de crear un algoritmo que se adapte a estar particularidades del código. Para poder entender nuestro código debemos de elaborar un diagrama de flujo básico, que se presenta a continuación.

5.1DIAGRAMA DE FLUJO



5.2 CÓDIGO IMPLEMENTADO

Conociendo el diagrama de flujo del apartado 4.3.1 podemos ir comentando como hemos ido realizando cada parte del mismo, incluyendo si fuera necesaria información adicional sobre el mismo. El siguiente código se ha implementado mediante la herramienta informática MATLAB. Siguiendo el orden impuesto por el diagrama anterior, hemos obtenido el bucle principal como se muestra en la Fig. 30.

```
while
(count<=iter max)
```

Fig. 30:Código del bucle principal.

Estos dos bucles son los principales y son los que usamos para recorrer el resto de vectores, que aparecerán en las siguientes secciones. Para generar un vecino aleatorio hemos usado el código de la Fig. 31.

```
for k=2:(D+1)

if (version_de==5)

      Rand=random('unid',NP);
     while((Rand==i)||(Rand==0))
       Rand=random('unid',NP);
     end

trial(1,k-1))=population(Rand,k)+SA*(population(Rand,k)-population(1,k));
```

Fig. 31: Código de generación de un nuevo candidato.

Como hemos indicado anteriormente, el SA debe generar vecinos aleatorios cerca del actual para evaluar si son mejores o no. En este punto nos encontramos con un gran problema, ya que nuestro SA debe usar un conjunto de soluciones finitas que hemos generado previamente y se almacenan en el vector *population*. La solución adoptada es mover el punto aleatorio del vector una parte proporcional de la diferencia de ese vector con la mejor solución obtenida. Esta solución ingeniosa nos permite movernos dentro del mapa y reajustar si fuera necesario el movimiento con la variable denominada SA (en honor

al nombre del algoritmo). Se ejecuta con la mejor solución debido a que de esta manera procuramos que los puntos converjan hacia la solución óptima. Debemos de movernos entre todos los puntos posteriormente para saber si este nuevo vecino es mejor que el punto que tenemos en cada parte del mapa arriba indicado. De lo contrario no moveremos los puntos lo suficiente para cubrir el mapa entero, o si nos limitamos a operar con el mejor candidato, caeremos en el mínimo en el que nos hallemos, sea local o global.

Con la variable k nos moveremos dentro de los vectores que contienen las soluciones propuestas, ya sea el vector *population* o el *trial*.

Posteriormente, generar un nuevo vecino lleva implícito que debe encontrarse en el mapa. No ha sido necesario incluirlo en el diagrama del apartado 4.3.1 y se ha decidido obviarlo, pero su implementación es necesaria. Comprobamos que cada vector del nuevo vecino propuesto se encuentra entre el máximo y el mínimo número de celdas del mapa. Una vez aceptado el vector como nuevo vecino, debemos comprobar su error frente a la medición del láser real con las funciones mostradas en la Fig. 32.

```
laser_est_from_trial=dist_est_2D(trial(1:3),map,mapmax,mapmin,err_dis,NUM
_MEASUREMENTS,SENSOR_RES,SENSOR_RANGE,T);

error_trial=fitness_2D(laser_real,laser_est_from_trial,version_fitness,NU
M_MEASUREMENTS);
```

Fig. 32: Código de las funciones de estimación láser y su error.

Por un lado, con la función dist_est_2D generamos la estimación del láser que usaremos para el cálculo del error mediante la función de coste. Dado que este código no es de mi autoría, lo añadiré en el anexo.

Por otro lado, hacemos uso de la función de costes que hayamos decidido para devolvernos el error. Como hemos explicado anteriormente, debemos pasarle los parámetros del láser real, los de nuestra estimación, la versión elegida y el número de medidas horizontales.

A continuación, siguiendo el orden establecido en el diagrama de flujo, debemos comparar el nuevo vecino con la solución actual. Para ello emplearemos el código de la Fig. 33.

En este fragmento, establecemos una primera condición en la que comparamos el error del nuevo candidato y el error del actual. Si resulta mejor el nuevo candidato, lo adoptamos como nueva solución de nuestro problema. Usamos un vector auxiliar llamado *pob_aux* para evitar realizar cambios en el conjunto de soluciones hasta que todas ellas hayan sido analizadas y evitar una pérdida de datos.

```
if(error_trial<population(i,1))
          for j=2:(D+1)
              pob_aux(i,j)=trial(1,j-1);
          end
          pob_aux(i,1)=error_trial;
else


        if rand<min(0,1/(1+exp((-error_trial+population(i,1))
        /population(i,1))))

              for j=2:(D+1)
              pob_aux(i,j)=trial(1,j-1);
              end
              pob_aux(i,1)=error_trial;

        else
              for j=1:(D+1)
                  pob_aux(i,j)=population(i,j);
        end
 end
```

Fig. 33:Código de selección. En él se comprará el nuevo candidato con la solución actual.

Si no resulta mejor, debemos pasarle la condición de aceptación descrita en la página - 34 - bajo el nombre de q. Esta condición describía la probabilidad de aceptar un candidato peor que el actual, por lo que es necesario generar un número aleatorio con la función propia para este fin de la herramienta MATLAB, llamada rand. Esta función genera un número al azar entre 0 y 1, lo que nos servirá para usar dicha probabilidad. Las variables de

temperatura y energía, como no puede ser de otra manera, cobran otro sentido dentro de nuestro problema particular adquiriendo la equivalencia de la Tabla 4.

Tabla 4: Equivalencias entre las variables del SA original y la solución propuesta.

Energía	Error
Temperatura	Pose del robot

Esta equivalencia no es un concepto inamovible, se debe a que en nuestras iteraciones no podemos ejecutar cambios en el error. Por tanto, no podemos valorar dicha variable como la temperatura del original. Esto nos limita a establecer la energía como el error, lo que, por otro lado, parece una solución factible ya que implica valorar si merece o no la pena moverse al siguiente estado según el coste que nos devuelve la función de costes. Lo cual nos lleva a valorar la temperatura como la pose del robot, pudiendo modificar dicha pose en cada iteración del algoritmo como se explicó en la página 56. La siguiente alteración de dicha condición surgió al intentar dividir el $\Delta error$ entre la pose, como se describe originalmente en el SA. Si ejecutamos dicha división al minimizar la temperatura, la pose del robot, debería modificar el valor de la probabilidad de aceptación lo que, como es lógico, no ocurre ya que la pose se compone de las coordenadas del robot y su orientación. Estas variables no son minimizables, pero sí el error. Por ese motivo re-articularemos la ecuación y probaremos su correcto funcionamiento empíricamente. Los resultados obtenidos fueron que el valor de la probabilidad rondaba el 0.5 en casos poco desfavorables y 0 en casos muy desfavorables. Cumpliendo así el criterio nombrado anteriormente que repito como recordatorio:

"*Si ΔE es positivo y T (que por las condiciones del algoritmo no puede bajar de cero) también lo es, esta probabilidad debe rondar entre 0 y 0.5.*"

Podemos concluir que hemos elaborado una adaptación válida para dicho criterio y podemos incorporarla a nuestro trabajo.

Para generar los outputs que nos piden, hemos de generar los mejores y peores errores que hemos obtenido en nuestras muestras de MCMC que hemos acabado de mejorar.

Para ello reutilizamos el código que usaba los métodos evolutivos para generarlos, y además nos mejoran positivamente la estructura de nuestras posibles soluciones ya que las ordena de mejor a peor solución, en función del error. Esto es muy útil para ver cuál es el mejor candidato y acercarnos a él, como necesitábamos al generar un nuevo candidato en el primer trozo de código. Y, además, no nos influye en la ejecución de nuestro código ya que nos movemos entre en espacio de muestras de manera aleatoria.

En el siguiente paso del algoritmo siguiendo el diagrama de flujo, debemos comprobar si hemos llegado al límite de iteraciones máximas. Condición que asegura la variable que mueve el bucle. Y si hemos llegado a la convergencia o a la solución óptima global.

Para ayudar a fomentar este criterio, movemos dos variables. Por un lado, SA que es nuestra adaptación del parámetro de *reannealing* (k en el algoritmo original). Y por otro El número de muestras tomadas en el espacio de posibles soluciones, reduciendo el espectro de posibilidades a los vecinos más inmediatos de la mejor solución. Estos valores tienen dos parámetros de límite que podemos modificar en una posterior mejora del código para favorecer una ejecución más eficiente que son respectivamente, el número de iteraciones y que el error del mejor y del peor candidato sean cero.

```
if count>100
    SA=0.3;
End


if error-error_max==0
    NP=10;
end
```

Fig. 34:Código perteneciente a las condiciones de reajuste del algoritmo.

Este fragmento de código se puede consultar en la Fig. 34. Arriba se muestra el parámetro de *reannealing*. Abajo se muestra la reducción del espacio de posibles soluciones.

Para favorecer la convergencia se usa una adaptación de la mejora propuesta para el método de DE llamada *discarding* o descarte. En esta propuesta, los peores candidatos se descartan para mejorar el espacio de soluciones levemente. Se ha creído conveniente aprovechar esta ventaja también para el SA como una nueva mejora ya implementada, que reduce el tiempo en resolver el problema de LG. De este modo descartamos las peores soluciones y las sustituimos por otros candidatos más cercanos al valor óptimo. Tras varias pruebas no ha sido posible concluir empíricamente un malfuncionamiento del algoritmo por el hecho de incluir este fragmento.

Presentaremos los resultados con el formato que se muestra en la Fig. 35. En él se muestra el número de iteraciones totales, el mejor y peor registro de errores en nuestro conjunto de soluciones, el error global, la ratio entre las estimaciones correctas y las totales necesarias para llevar a cabo el resto de resultados, y por último y más importante la pose real.

```
It: 5.000000 Best 97.345603 Worst: 1373.359151 Global: 156814.636277 Best/Measurements: 1.595830
Position: x y theta 866.334624 59.374547 0.000000 .
```

Fig. 35: Ejemplo de resultados obtenidos por pantalla cada 5 iteraciones.

6. RESULTADOS OBTENIDOS

A continuación, realizaremos varias pruebas para realizar una evaluación del funcionamiento del código anteriormente descrito en la búsqueda del óptimo. Intentaremos sacar los errores más significativos y buscar una solución si fuera posible. Para ello contaremos con varios ejemplos de ejecución muy detallados, y otros ejemplos complementarios para poder esclarecer errores y asegurar el correcto funcionamiento del algoritmo.

El formato de dichas pruebas será siempre el mismo para poder comparar todos los resultados. Usaremos la norma L2 como función de costes y el mapa total (Fig. 25) como banco de pruebas. Los resultados aparecerán en tres gráficas idénticas que contienen el mapa seleccionado. En la gráfica número 3 veremos una evolución de nuestro algoritmo hacia la búsqueda de la posición real. En las dos primeras obtendremos los resultados entre la estimación de la pose del robot y la solución gráfica de las lecturas del láser, y la real.

6.1 EJEMPLO 1

En este ejemplo vamos a probar una zona en la que haya menos probabilidades de confundir las lecturas del láser, e intentar que no sea exacta a otra zona del mapa. Probaremos con la pose, dando los parámetros reales del robot como (θ, x, y), (0, 50, 60) y 120 iteraciones máximas. Tras resolver el problema en 7.06s, hemos obtenido los resultados mostrados en la Fig. 36.

```
It: 120.000000 Best 19.770492 Worst: 19.848361 Global: 3954.254098 Best/Measurements: 0.324106
Position: x y theta 52.607750 61.196088 360.000000 .
Elapsed time is 7.064478 seconds.

Robot real pose  (x y theta) 50.000000 60.000000 0.000000
Estimated pose given by the GL filter (x y theta) 52.607750 61.196088 360.000000
The position error is 34.714528 cm and the orientation error is 360.000000 grados
```

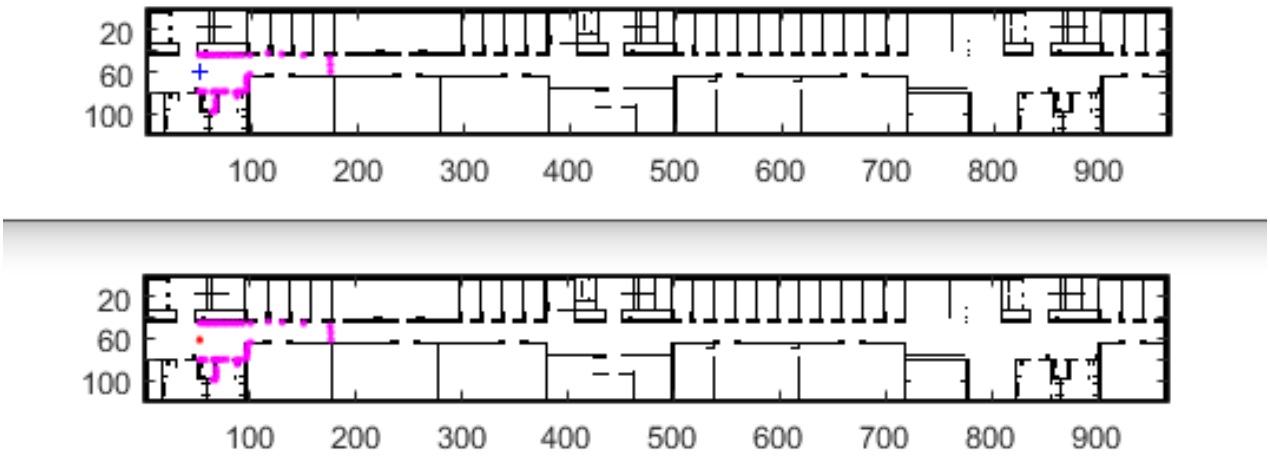
Fig. 36: Resultados obtenidos tras la última iteración del ejemplo 1 de SA.

Podemos observar que existe cierto error en la posición, mientras que en la orientación no ha ninguno (360º = 0º). El error de posición es de 34.71 cm aproximadamente lo cual me hace pensar en una primera conclusión que no ha sido capaz de hallar el óptimo global, pero ha resultado con cierta precisión, más o menos certero en el resultado.

Si observamos la premisa de la posición alcanzada por el algoritmo en la Fig. 36, podemos deducir su posición como se aprecia en la Fig. 37.

Fig. 37: Resultado gráfico de convergencia del SA tras la prueba número 1.

Dada la escala, observamos que ha llegado al punto óptimo mejor de lo que aparentaban las cifras anteriores. Pero para ser más exactos, nuestro punto de gran interés es ver gráficamente si ha conseguido dar una estimación realmente útil al robot de su pose. Y esto lo descubriremos con las dos primeras gráficas del mapa, que corresponden a la Fig. 38.



Arriba, con una cruz azul, mostramos la pose real del robot y en color rosa las lecturas reales de su láser. Abajo, con el punto rojo, podemos observar la posición y las lecturas estimadas del láser mediante el algoritmo de SA. Por lo que ahora sí, podemos concluir que el primer test ha sido satisfactorio, y ha logrado hacer una buena estimación de la pose del robot en 120 iteraciones. Cabe destacar que a partir de la iteración 100 reajustamos el parámetro de reannealing (SA). Por otro lado, muestra un único punto ya que las soluciones candidato son reducidas cuando se alcanza el criterio de convergencia.

Si estudiamos más en detalle las soluciones podemos observar que mucho antes de las 120 iteraciones ya habíamos hallado dicha solución y las siguientes iteraciones no han sido necesarias.

Fig. 38:Resultado gráfico de la pose del robot tras la ejecución del SA en el ejemplo 1.

```
It: 20.000000 Best 49.508197 Worst: 460.073382 Global: 55413.663834 Best/Measurements: 0.811610
Position: x y theta 52.607750 58.142243 360.000000 .

It: 25.000000 Best 20.680328 Worst: 342.658129 Global: 33596.265921 Best/Measurements: 0.339022
Position: x y theta 52.607750 61.153925 360.000000 .

It: 30.000000 Best 19.770492 Worst: 206.618852 Global: 15234.826603 Best/Measurements: 0.324106
Position: x y theta 52.607750 61.196088 360.000000 .

It: 35.000000 Best 19.770492 Worst: 70.237705 Global: 6259.098988 Best/Measurements: 0.324106
Position: x y theta 52.607750 61.196088 360.000000 .

It: 40.000000 Best 19.770492 Worst: 27.635246 Global: 4169.401639 Best/Measurements: 0.324106
Position: x y theta 52.607750 61.196088 360.000000 .
```

Fig. 39: Detalle de las soluciones aportadas por la primera prueba del SA. Iteraciones 20-40.

Fijémonos bien en las iteraciones de la Fig. 39. En primer lugar, observaremos que desde la iteración número 30 la mejor solución no se ha modificado. Esto puede ocurrir ya que nos movemos por el espacio de posibles soluciones de manera aleatoria. La consecuencia directa repercute en el peor error, dado que nos aproximamos (por el descarte realizado en el código) a la mejor solución eliminando las peores, podemos observar como este valor decrece rápidamente una vez hallado el punto óptimo. Esto disminuye la ratio de buenas

aproximaciones / intentos *(best/Measurements*). Por tanto, para la siguiente prueba emplearemos un límite más bajo de iteraciones máximas y en tercer lugar modificaremos el código para facilitar dicha aproximación.

6.2 EJEMPLO 2

Como hemos comentado en el ejemplo anterior, este nuevo test lo vamos a realizar con 60 iteraciones máximas y la misma pose. Obteniendo los siguientes resultados:

```
It: 60.000000 Best 15.046694 Worst: 16.645054 Global: 3019.014983 Best/Measurements: 0.246667
Position: x y theta 856.126229 60.287989 0.000000 .
Elapsed time is 5.631380 seconds.

Robot real pose  (x y theta) 50.000000 60.000000 0.000000
Estimated pose given by the GL filter (x y theta) 856.126229 60.287989 0.000000
The position error is 9754.127989 cm and the orientation error is 0.000000 grados
```

Fig. 40: Resultados obtenidos tras la última iteración del ejemplo 2 de SA.

En este ejemplo, el mejor error y el peor son incluso mejores que en el primero, con la mitad de iteraciones. Pero si nos fijamos en la pose estimada vemos un valor de error de posición de 9754.13cm, debemos deducir como podemos explicar este resultado. Pues bien, vamos a observar las tres gráficas que nos aporta el código para tratar de buscar una respuesta lógica:

Fig. 41: Resultado gráfico de convergencia del SA tras la prueba número 2.

Como podemos ver en la Fig. 41, no ha convergido en el punto óptimo, sino en la otra punta del mapa. Esto se debe a que ambas posiciones son idénticas y es un error de estimación que cabía esperar como hemos explicado anteriormente. Puesto que no tenemos más detalle que una medida láser, si el espacio que debemos simular y el simulado son idénticos, podemos obtener un error en la localización (de casi 100m), mientras que en

la estimación del error de la medida no tenemos apenas desviación (15cm). Este fenómeno se ve mucho más claro en la Fig. 42, dónde ambas lecturas del láser (arriba real y abajo estimada, de color rosa) son idénticas a pesar de realizarse en distintos lugares del espacio.

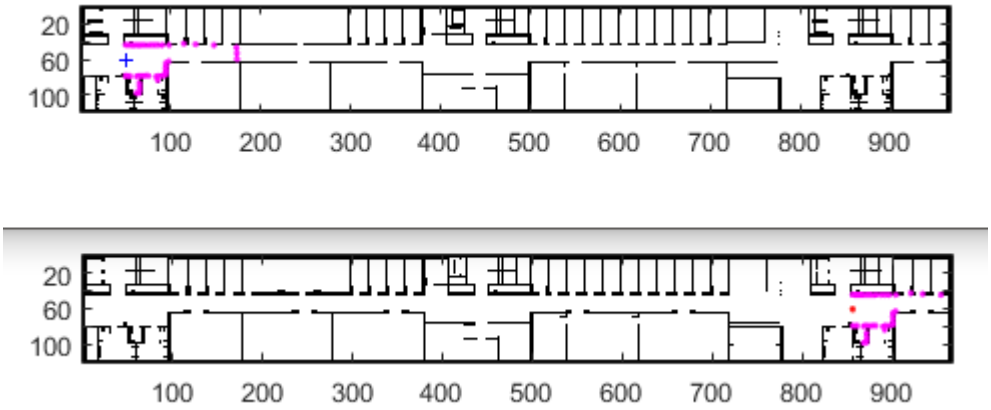


Fig. 42: Resultado gráfico de la pose del robot tras la ejecución del SA en el ejemplo 2.

Si nos fijamos en los detalles de la ejecución de cada bucle, podemos observar en un temprano estancamiento en la iteración 30 de la estimación más próxima a la solución óptima. Lo cual podría indicar no haber alcanzado en mínimo global Fig. 41Fig. 43.

Con todo este conocimiento en nuestro poder, podemos concluir que nos hemos estancado quizás en un óptimo local, o bien que es un error insalvable. Si fuera la primera suposición, debemos darle más iteraciones al algoritmo para que pueda salir de dicho máximo. Si por el contrario nuestra teoría fuera correcta, podemos asumir el error. Lo iremos observando en el resto de experimentos.

```
It: 25.000000 Best 39.784399 Worst: 311.306821 Global: 39062.985124 Best/Measurements: 0.652203
Position: x y theta 856.126229 58.410726 360.000000 .

It: 30.000000 Best 15.046694 Worst: 212.627049 Global: 20082.260621 Best/Measurements: 0.246667
Position: x y theta 856.126229 60.287989 0.000000 .
```

Fig. 43: Detalle de las soluciones aportadas por la segunda prueba del SA. Iteraciones 25-30.

6.3 EJEMPLO 3

Debido a la sucesión de los acontecimientos que han tenido lugar en el ejemplo 2, vamos a cambiar nuestro tercer ejemplo, ya que de acortar de nuevo la ejecución del algoritmo puede llevarnos a una temprana solución no global. Por tanto, vamos a concluir este paquete de pruebas moviendo la pose a una diferente más difícil de distinguir, pero con bastantes iteraciones, 120. De esta manera podremos confirmar o desmentir las suposiciones que hemos ido planteando a lo largo de los ejemplos anteriores. Probaremos la siguiente pose (20, 200, 100). Esta pose posee una orientación diferente que dificultara el reconocimiento de las esquinas y además hemos optado por un punto dentro de uno de los laboratorios idénticos entre sí. Sea cual fuere el resultado, podrá darnos los suficientes datos para sacar una conclusión firme. Y sin más dilación estos son los resultados obtenidos:

```
It: 120.000000 Best 10.311475 Worst: 14.836066 Global: 2071.344262 Best/Measurements: 0.169041
Position: x y theta 304.037069 103.475735 17.189027 .
Elapsed time is 11.277809 seconds.

Robot real pose  (x y theta) 200.000000 100.000000 20.000000
Estimated pose given by the GL filter (x y theta) 304.037069 103.475735 17.189027
The position error is 1259.550866 cm and the orientation error is 2.810973 grados
```

Fig. 44: Resultados obtenidos tras la última iteración del ejemplo 3 de SA.

Los resultados no han sido los deseados, pero nos permiten sacar conclusiones.

Por un lado 120 iteraciones sabemos que son más que suficientes para llegar al óptimo global. El error de la estimación (10.31cm) es el más bajo de los tres ejemplos, mientras que el error de posición es el más alto (120m). Lo que nos lleva a pensar que en este caso se ha vuelto a equivocar y debemos saber el porqué de este error, que analizaremos con la Fig. 45.

Fig. 45: Resultado gráfico de convergencia del SA tras la prueba número 3.

En ella podemos observar que el error es el esperado, y si vamos más allá, entre todas las habitaciones idénticas, se ha equivocado en la más cercana. Lo que indica que no halla el óptimo global, pero debo intuir que se trata por no poseer más datos de la posición que la medición laser. Ya que el error de estimación es el más bajo hasta la fecha debemos comprobar con la Fig. 46 que las medidas láser, tanto del real como de la estimación, son idénticas y de ahí el poco error de estimación. Por lo que podemos concluir, sin temor a equivocarnos, que sin más datos de orientación el robot nunca será capaz de distinguir entre dos habitaciones idénticas.

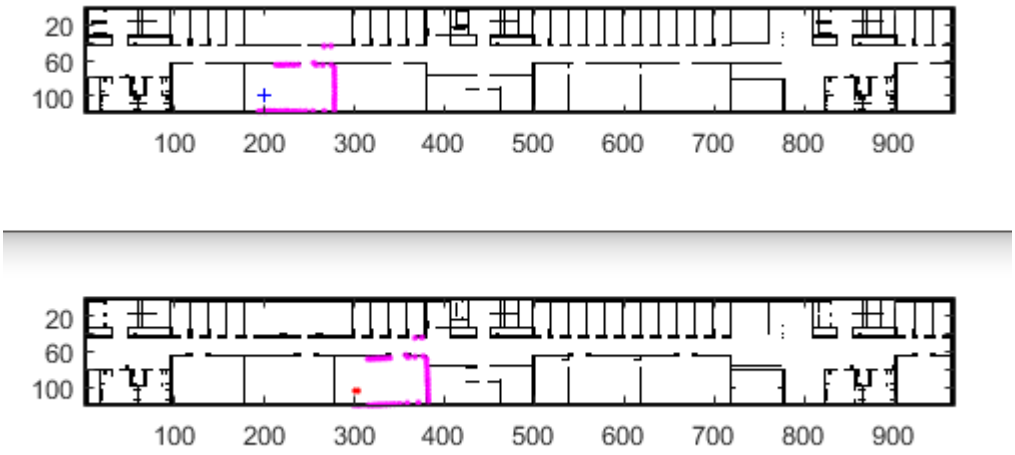


Fig. 46: Resultado gráfico de la pose del robot tras la ejecución del SA en el ejemplo 3.

```
It: 60.000000 Best 14.836066 Worst: 78.934426 Global: 8239.180328 Best/Measurements: 0.243214
Position: x y theta 300.510729 103.475735 17.189027 .

It: 65.000000 Best 10.311475 Worst: 42.872951 Global: 4330.909836 Best/Measurements: 0.169041
Position: x y theta 304.037069 103.475735 17.189027 .

It: 70.000000 Best 10.311475 Worst: 14.836066 Global: 2836.000000 Best/Measurements: 0.169041
Position: x y theta 304.037069 103.475735 17.189027 .
```

Fig. 47: Detalle de las soluciones aportadas por la tercera prueba del SA. Iteraciones 60-70.

Tras fijarnos en los detalles ha terminado de resolver el mejor caso en la iteración número 65. Lo que corrobora que ha obtenido mucho tiempo excedente para salir del mínimo local y no lo ha hecho, por tanto, el motivo de este error es el ya deducido anteriormente de las habitaciones idénticas.

6.4 EJEMPLOS COMPLEMENTARIOS

Hemos querido incluir otros ejemplos complementarios para demostrar de una manera cómoda y rápida el buen funcionamiento del algoritmo, y las conclusiones obtenidas en los anteriores apartados. Tras probar nuestra teoría en otros mapas y con otras coordenadas obtenemos estos resultados:

En este caso Fig. 48, no consigue (tras 120 iteraciones) distinguir cual es la pose real, obteniendo 3 candidatos a solución óptima que no han podido ser descartados. Hecho que

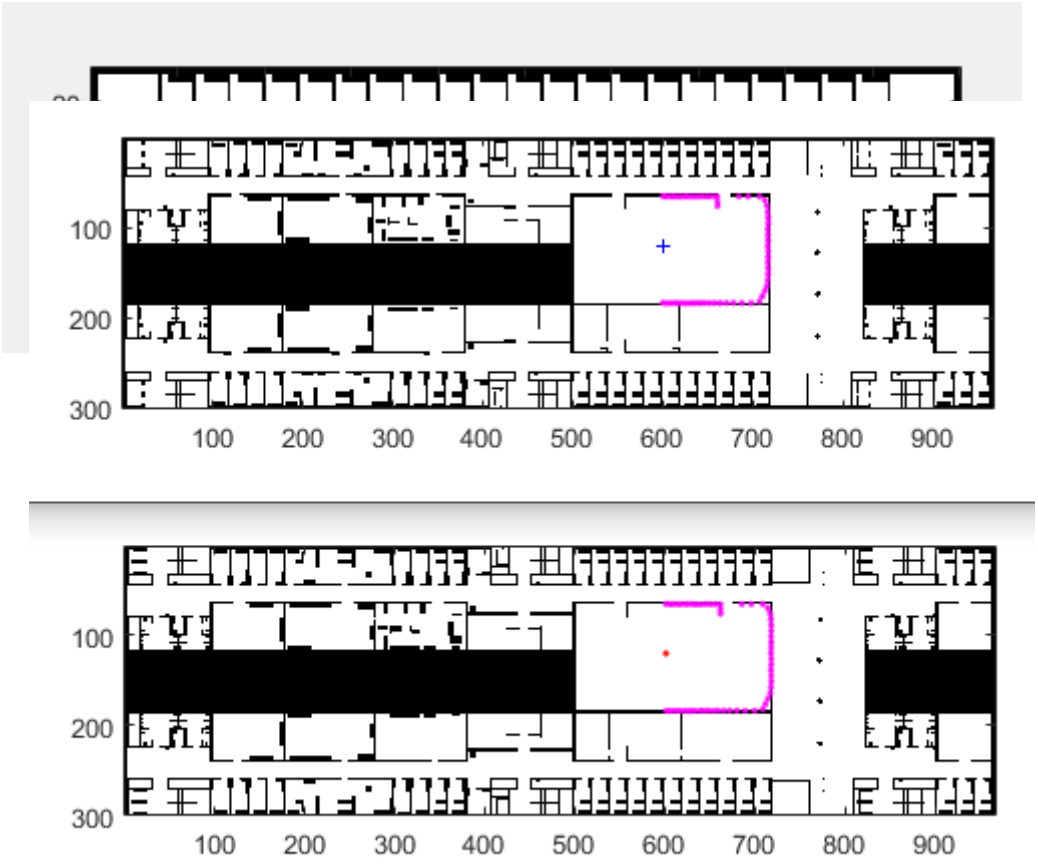


Fig. 49: Prueba en mapa enorme, donde acierta sin problemas el SA.

corrobora totalmente nuestras conclusiones previas.

En la figura anterior, vemos como probándolo en un mapa enorme, en una zona de no conflicto halla la pose real sin ningún problema. Este hecho nos deja con un código robusto que en la aplicación real no compondrá ningún problema menor, a pesar de que en estos casos teóricos si lo haga. Halla el óptimo con una precisión absoluta como se puede comprobar en la siguiente figura:

7. CONCLUSIÓN

Para terminar, trataremos de dar sentido a todo el trabajo elaborando una conclusión de los ejemplos y procesos que hemos llevado a lo largo del mismo.

Por un lado, hemos elaborado un código que adapta la solución generalista del SA a nuestro problema particular. Adaptando las variables que intervienen en el proceso, tanto la temperatura, como la energía, como el parámetro de reannealing, entre otros. Hemos ido demostrando como esas adaptaciones tenían sentido a lo largo de la descripción del código para terminar dando ejemplos prácticos de su funcionamiento. Por esta parte, podemos concluir que el código ha sido un éxito y hemos logrado resolver el problema de LG mediante el método de SA, en nuestra particular adaptación. Elaborando un código complementario a los ya existentes dentro del programa, que no interfiere en el funcionamiento de los mismos y cumple todas las premisas impuestas tanto de programación como de conocimientos teóricos de los MCMC y del SA.

```
It: 120.000000 Best 0.556853 Worst: 0.556853 Global: 111.370606 Best/Measurements: 0.009129
Position: x y theta 601.242764 120.288217 359.877178 .
Elapsed time is 12.383452 seconds.

Robot real pose  (x y theta) 600.000000 120.000000 0.000000
Estimated pose given by the GL filter (x y theta) 601.242764 120.288217 359.877178
The position error is 15.436544 cm and the orientation error is 359.877178 grados
```

Fig. 50: Resultados obtenidos tras la última iteración de la prueba del mapa grande mediante el método SA.

Por otro lado, hemos establecido unas premisas empíricas en el primer ejemplo que nos daban problemas y necesitábamos de su estudio aun funcionando con normalidad dicho código. Tras un profundo análisis, hemos visto como ha sido imposible distinguir entre

dos habitaciones idénticas en un primer reconocimiento del láser. Problema que no tendremos en la aplicación real ya que en una segunda lectura podremos concretar fácilmente la pose exacta del robot, que no pudimos obtener sin más datos cognitivos. A priori, este problema parecía insalvable, pero hemos podido estudiar que en casos particulares sin réplicas exactas el algoritmo funciona como debe. El resto de casos en una situación real también es poco probable que los encontremos puesto que encontraremos objetos móviles, mobiliario, y demás objetos que nos ayudaran en la tarea de distinción de LG. Por lo que por esta parte podemos concluir que el algoritmo funciona correctamente y es viable para la aplicación real. Obteniendo unos tiempos de respuesta de pocos segundos, claramente mejorables mediante la optimización y extracción del código de su integración dentro del programa completo.

Para terminar, podemos sacar una conclusión global. Y es que el algoritmo ha cumplido el objetivo principal del trabajo y ha sido un éxito. A continuación, propondremos un par de mejoras futuras con las que despediremos la parte de investigación de nuestro trabajo.

7.1 MEJORAS FUTURAS

En este apartado vamos a tratar de proponer alguna mejora que se podrá realizar en una versión beta del código.

En primer lugar y como ya hemos introducido, debemos aislar nuestro código del programa general y su función de costes óptima. Esto reducirá notablemente los tiempos de ejecución y parametrización del problema, que una vez aplicado, serán aún más cortos no siendo necesaria la interacción humana.

En segundo lugar, debemos hallar el punto óptimo de resolución y fijar unas iteraciones máximas de un modo más exacto y experimental. Esto se ha tratado de realizar en el trabajo sin éxito debido a los problemas presentes en la ejecución teórica anteriormente descritos.

Las siguientes mejoras resultan de mejorar la precisión del algoritmo. Para ello debemos proponer:

En tercer lugar, redefinir el límite de acercamiento al valor óptimo. Este ajuste fino debe realizarse de manera empírica obteniendo un valor óptimo, que aumentará el rendimiento y disminuirá el tiempo que tardará en alcanzar la solución óptima global.

En cuarto lugar, incluir la posibilidad de reiniciar el espacio de estados si tras un número adecuado de iteraciones no hemos sido capaces de llegar a la solución óptima. Evitando de manera eficaz caer en una solución local.

En quinto lugar, ajustar de manera empírica el parámetro de reannealing SA para optimizar el rendimiento y el tiempo de carga del algoritmo.

Y en último lugar, y de la misma manera que el punto anterior reajustar la probabilidad de aceptación g probando caso a caso cuando es necesario y no realiza suficientes salidas de la rama secuencial del algoritmo.

El siguiente paso por tanto debe ser poner en marcha el algoritmo y generar una base de datos de todas las LG realizadas para poder pulir el código.

Bibliografía

[1] Tecnalia, «Tecnalia,» [En línea]. Available: http://www.tecnalia.com/es/industria-transporte/eventos/industria-40-la-fabrica-inteligente.htm. [Último acceso: 13 09 2016].

[2] RAE, «Diccionario de la lengua española,» 23 09 2016. [En línea]. Available: http://dle.rae.es/?id=WYTm4uf.

[3] wikimedia, «wikimedia.org,» [En línea]. Available: https://upload.wikimedia.org/wikipedia/commons/thumb/f/f5/Bayes'_Theorem_2D.svg/220px-Bayes'_Theorem_2D.svg.png.

[4] N. Vahdat, Mobile robot global localization using differential evolutionand particle swarm optimization, in : Proceedings of the Congresson Evolutionary Computatio, 2007.

[5] M. Lisowski, Differential evoution approach to the localization problem for mobilerobots,Master'sThesis., Dinamarca, 2009.

[6] R. Mirkhania, A novel efficient algorithm for mobile robot localization, Robot.Auton.Syst., 2013.

[7] V. Yepes, Optimización heurística económica aplicada a las redes de transporte del tipo VRPTW, 2002.

[8] J. Arranz de la Peña y A. Parra Truyol, «uc3m,» 2007. [En línea]. Available: http://www.it.uc3m.es/jvillena/irc/practicas/06-07/05.pdf. [Último acceso: 20 09 2016].

[9] M. Muñoz, «uc3m,» [En línea]. Available: http://e-archivo.uc3m.es/bitstream/handle/10016/2377/Tesis_Mendez_Munoz.pdf?sequence=6. [Último acceso: 20 09 2016].

[10] M. Gisbert Schneider, «Voyages to the (un)known: adaptive design of bioactive compounds,» [En línea]. Available: http://www.cell.com/trends/biotechnology/fulltext/S0167-7799(08)00259-X. [Último acceso: 20 09 2016].

[11] S. Kirkpatrick, Optimization by Simulated Annealing, 1983.

[12] V. Černý, Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm, 1985.

[13] H. José Alberto y S. Pablo , «uc3m,» [En línea]. Available: http://www.it.uc3m.es/pablo/teoria-colas/. [Último acceso: 21 09 2016].

[14] F. Hartig, «R-bloggers,» [En línea]. Available: https://www.r-bloggers.com/a-simple-metropolis-hastings-mcmc-in-r/. [Último acceso: 21 09 2016].

[15] UPV, «web de la Universidad Politécnica de Valencia.,» [En línea]. Available: http://www.upv.es/materiales/Fcm/Fcm04/Im%E1genes/Fig4_52.jpg. [Último acceso: 15 09 2016].

[16] UPV, «web de la Universidad Politécnica de Valencia,» [En línea]. Available: http://www.upv.es/materiales/Fcm/Fcm04/Im%E1genes/Fig4_53.jpg. [Último acceso: 15 09 2016].

[17] UPV, «web de la Universidad Politécnica de Valencia,» [En línea]. Available: http://www.upv.es/materiales/Fcm/Fcm04/Im%E1genes/Fig4_59.jpg. [Último acceso: 15 09 2016].

[18] Matlab, «Mathworks.com,» Mathworks, [En línea]. Available: http://es.mathworks.com/help/gads/how-simulated-annealing-works.html. [Último acceso: 21 09 2016].

[19] M. T. Abad, «Universidad Politécnica de Cataluña,» [En línea]. Available: http://www.cs.upc.edu/~mabad/IA/SIMULATED%20ANNEALING.pdf. [Último acceso: 21 09 2016].

[20] W. William, J. Cook, Combinatorial Optimization, 1997.

[21] V. Pierluigi Crescenzi, A Compendium of NP Optimization Problems.

[22] S. Christos H. Papadimitriou, Combinatorial Optimization: Algorithms and Complexity, 1998.

[23] H. Garfinkel, The Traveling Salesman Problem: A Guide Tour of Combinatorial Optimization, 1985.

[24] M. GRÖTSCHEL, Combinatorial Optimization: A Survey, 1993.

[25] V. Yepes, «Gráfica de procedimiento heurístico,» UPV, 2009.

[26] Universidad Politécnica de Valencia, «Departamento de materiales de la UPV,» [En línea]. Available: http://www.upv.es/materiales/Fcm/Fcm04/pfcm4_4_2.html. [Último acceso: 15 09 2016].

[27] S. H. AVNER, Introducción a la Metalurgia Física.

[28] R. E., Principios de Metalurgia Física, McGraw−Hill.

[29] C. P. Orcajo, «Universidad Autónoma de Madrid,» [En línea]. Available: https://www.uam.es/personal_pdi/ciencias/carlosp/html/pid/montecarlo.html. [Último acceso: 18 09 2016].

[30] J. M. Marín, «UC3M,» [En línea]. Available: http://halweb.uc3m.es/esp/Personal/personas/jmmarin/esp/PEst/tema4pe.pdf. [Último acceso: 19 09 2016].

[31] G. P. Basharin, A. N. Langville y V. A. Naumov, The life and Work of A. A. Markov, 2004.

[32] M. Lisowski, Differential evoution approach to the localization problem for mobilerobots, 2009.

[33] D. Solis, «slide share,» 28 11 2015. [En línea]. Available: http://es.slideshare.net/dsolis/calibracin-del-modelo-heston-usando-evolucin-diferencial. [Último acceso: 19 09 2016].

[34] M. J. M., «Universidad Carlos III de Madrid,» [En línea]. Available: http://halweb.uc3m.es/esp/Personal/personas/jmmarin/esp/Bayes/tema5bayes.pdf. [Último acceso: 21 09 2016].

[35] S. G. Reid, «Turing Finance,» [En línea]. Available: http://www.turingfinance.com/simulated-annealing-for-portfolio-optimization/. [Último acceso: 21 09 2016].

[36] D. Bertsimas y J. Tsitsiklis, «MIT,» [En línea]. Available: http://www.mit.edu/~dbertsim/papers/Optimization/Simulated%20annealing.pdf. [Último acceso: 21 09 2016].

ANEXOS

A. CÓDIGO MATLAB MC CÁLCULO DE PI

```
rand('state',123)
n = 5000;
J=rand(n,2)*2-1;
k=sum(J.^2,2)<1;
m=sum(k);
piestimate=4*m/n;
pi;
figure('color',[1,1,1]);
plot(J(k,1),J(k,2),'b.');
axis equal;
```

B. PRESUPUESTO

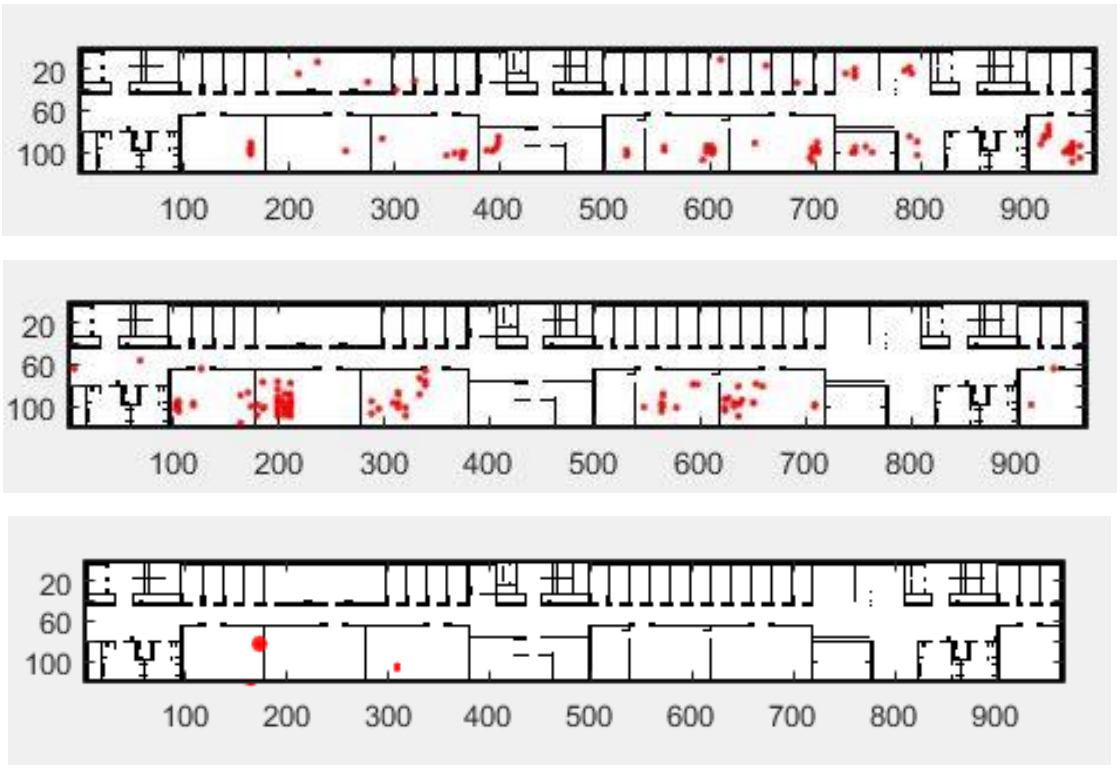
A continuación, se estimará un presupuesto de la ejecución del proyecto. Dado que ha sido un proyecto de software, solo tendremos en cuenta recursos humanos en su elaboración y despreciaremos cualquier recurso material como pueda ser la luz o el internet.

Código	Medición	Descripción	Medida	Precio Ud. (€)	Total (€)
1.1	Horas	**Dedicación del tutor.** Número de horas dedicadas a la resolución y corrección del trabajo hasta su forma final. En horas.	40	25	1000 €
1.2	Horas	**Dedicación del alumno.** Número de horas dedicadas a la elaboración y corrección del trabajo hasta su forma final. En horas.	300	15	4500 €
				Total:	5500 €

C. ETAPAS DE LA RESOLUCIÓN DEL SA

Apoyo gráfico del proceso:

En una primera aproximación, el algoritmo se va almacenando en los óptimos locales. Y descartando los candidatos menos óptimos.



Hasta que logra alcanzar un máximo local como hemos demostrado en la sección 0.

D.CÓDIGO DE LA FUNCIÓN DE COSTE

```
% L2 Norm. (Sum of the squared errors).
if (version_fitness==1)
    error=sum((laser_real - laser_estimate).^2)/NUM_MEASUREMENTS;
end
```

E. CÓDIGO DE LA FUNCIÓN PRINCIPAL

```
clc

% The GL module works with different maps (placed in the maps folder).
The
% user can choose one of them.
[map_real, map_known]=map_loading;
% The transposed map is used to estimate the sensor measurements.
map_known_tr=int8(map_known');
map_real_tr=int8(map_real');
% The map limits are computed
[m,n]=size(map_known_tr);
mapmax=[m,n,360];
mapmin=[1,1,0];

% The map is displayed in two different figures.
figure(1);
imagesc(map_real,'CDataMapping','scaled')
set(gcf,'Color','white');
set(gcf,'BackingStore','off');
set(gca,'DataAspectRatio',[1 1 1]);
colormap('gray')
hold on

figure(2);
imagesc(map_known,'CDataMapping','scaled')
set(gcf,'Color','white');
set(gcf,'BackingStore','off');
set(gca,'DataAspectRatio',[1 1 1]);
colormap('gray')
hold on

%-------------------------------------------------------------------------
--
%Initialization parameters:
NUM_MEASUREMENTS=61;        % Number of horizontal measurements in a
scan.
SENSOR_RES=3*pi/180;        % Laser sensor angular resolution (radians)
CELL_SIZE=0.121;            % Cell size, in m.
T=0.5;                     % Translation constant. To estimate laser
beams
                            % in order to generate a laser scan,
                            % dist_est_2D considers increments of T units
                            % in the map.
```

```
SENSOR_RANGE=15/CELL_SIZE;  % Laser sensor range
D=3;                        % DE Number of Chromosomes.
F=0.8;                      % Differential variations factor (mutation)
CR=0.5;                     % Crossover constant
% Variables introduced via keyboard
[position, err_dis, iter_max]=initialization(mapmin,mapmax,map_real');

%-------------------------------------------------------------------------
--
% The laser scan is generated according to the robot's true pose in the
% known map.
laser_real=dist_est_rob_2D(position,map_real_tr,mapmax,mapmin,err_dis,NUM
_MEASUREMENTS,SENSOR_RES,SENSOR_RANGE,T);

%-------------------------------------------------------------------------
--
%   Initialization of the population size (NP). Two options:
%                1: Initialized by function init_NP
%                Else: Fixed size given by code.
NP_opt=2;
if NP_opt==1

NP=init_NP(laser_real,err_dis,mapmax,SENSOR_RES,NUM_MEASUREMENTS,CELL_SIZ
E);
else
    NP=200;
end
NP=round(NP);
fprintf(1,'\n Population size: %i \n',NP);

%-------------------------------------------------------------------------
--
% Different options for the GL algorithm can be selected via keyboard:
%  - DE Core Options:
%    1) Random Mutation, with Thresholding and Discarding (Default).
%    2) Basic version, Random Mutation, without Thresholding, Discarding.
%    3) Mutation from Best candidate, with Thresholding and Discarding.
%    4) Random Mutation, with Thresholding and Discarding, NP is
%    drastically reduced (tracking) after convergence.
%    5) Simulated Annealing.
version_de=input('\ \n Introduce the DE version that you want to apply:
\n 1) Random Mutation, with Thresholding and Discarding. \n 2) Basic
version, Random Mutation, without Thresholding, Discarding. \n 3)
Mutation from Best candidate, with Thresholding and Discarding. \n 4)
Random Mutation, with Thresholding and Discarding, NP reduced (tracking)
after convergence. \n 5)Simulated Annealing. \n');
if isempty(version_de),
    version_de=1;
    fprintf(1,'\n \t Option 1 by default. \n');
end
%  - Fitness Function Options:
%    1) L2 Norm. Sum of the squared errors (Default).
%    2) L1 Norm. Sum of the absolute values of the error (Mahalanobis).
%    3) Kullback-Leibler Divergence based.
%    4) Density Power Divergence based.
%    5) Hellinger Distance based.
```

```
%     6) L2 Norm from Probability Distributions
%     7) L(Variable Exponent) Norm from Probability Distributions
%     8) Generalized Kullback-Leibler Divergence based.
%     9) Itakura-Saito Divergence based.
%     10) Jensen-Shannon Divergence based.
version_fitness=input('\ \n Introduce the Fitness Function that you want
to apply: \n 1) L2 Norm. Sum of the squared errors (Default). \n 2) L1
Norm. Sum of the absolute values of the error. \n 3) Kullback-Leibler
Divergence based. \n 4) Density Power Divergence based. \n 5) Hellinger
Distance based. \n 6) L2 Norm from Probability Distributions \n 7)
L(Variable Exponent) Norm from Probability Distributions. \n 8)
Generalized Kullback-Leibler Divergence based. \n 9) Itakura-Saito
Divergence based. \n 10) Jensen-Shannon Divergence based. \n');
if isempty(version_fitness),
    version_fitness=1;
    fprintf(1,'\n \t Option 1 by default. \n');
end

%-------------------------------------------------------------------------
--
% The initial population is randomly generated to cover the whole map.
population=initiate_pop(mapmin,mapmax,NP,D);

%-------------------------------------------------------------------------
--
% Some parame change for the DPD and HC function, extra D is needed, with
% limits.
if (version_fitness==4) || (version_fitness==7)
    population=[population rand(NP,1)]; % Extra column with alpha
    D=4;
    mapmin=[mapmin 0.001]; % Lower limit for alpha is 0
    mapmax=[mapmax 1]; % Upper limit for alpha is 1
end

%-------------------------------------------------------------------------
--

% The robot motion simulation starts. In a single step, the robot tries
to
% locate itself. After convergence, robot motion is allowed until an 'f'
% is introduced in dir_disp. In this case, the GL module ends its
% execution.
steps=0;
dir_disp=' ';
while (dir_disp~='f')

        fprintf(1,'\n Robot real pose  (x y theta) %f %f %f
\n',position(1),position(2),position(3));
        tic
        % The DE-based GL filter is called

[bestmem,error,population,F,NP]=alg_genet_2D(version_de,version_fitness,l
aser_real,map_known_tr,population,iter_max,mapmax,mapmin,err_dis,NUM_MEAS
UREMENTS,SENSOR_RES,NP,D,F,CR,SENSOR_RANGE,T);
        toc
```

```
        fprintf(1,'\n Robot real pose  (x y theta) %f %f %f
%f:\n',position(1),position(2),position(3));
        fprintf(1,'\n Estimated pose given by the GL filter (x y theta)
%f %f %f %f\n',bestmem(2),bestmem(3),bestmem(4));

        % The error between real pose and estimate is computed
        poserror=12.1*sqrt((position(1)-bestmem(2))^2+(position(2)-
bestmem(3))^2);
        orierror=abs(position(3)-bestmem(4));
        fprintf(1,'\n The position error is %f cm and the orientation
error is %f grados\n',poserror,orierror);

        %-----------------------------------------------------------------
        % Robot motion is allowed (after convergence) via keyboard
        MOTION_TRANSL_RES=3;  % translation motion resolution in a step
(cells)
        MOTION_ORIENT_RES=5;  % angular motion resolution in a step
(degrees)
        disp=zeros(3);     % Vector that contains the displacements
        dir_disp=input('\n Introduzce the movement direction: \n','s');
        if isempty(dir_disp)
            fprintf(1,'\n The default movement is zero \n');
        end
        if (dir_disp=='k')
            disp(1)=+MOTION_TRANSL_RES;
        elseif (dir_disp=='j')
            disp(1)=-MOTION_TRANSL_RES;
        elseif (dir_disp=='m')
            disp(2)=-MOTION_TRANSL_RES;
        elseif (dir_disp=='i')
            disp(2)=+MOTION_TRANSL_RES;
        elseif (dir_disp=='h')
            disp(3)=MOTION_ORIENT_RES;  %el angulo esta en radianes
        elseif (dir_disp=='l')
            disp(3)=MOTION_ORIENT_RES;
        end

        % The best solution is displaced according to the robot's motion.
        bestmem(2)=bestmem(2)+disp(1);
        bestmem(3)=bestmem(3)+disp(2);
        bestmem(4)=bestmem(4)+disp(3);

        % The real pose is displaced according to the robot's mootion,
but
        % including a Normally distributed error with mean given by the
        % coordinate of disp and standard deviation equal to
        % disp*MOTION_ERROR.
        MOTION_ERROR=0.03;      % Motion error standard dev, over disp.
        position(1)=position(1)+disp(1)*(1+MOTION_ERROR*randn(1));
        position(2)=position(2)+disp(2)*(1+MOTION_ERROR*randn(1));
        position(3)=position(3)+disp(3)*(1+MOTION_ERROR*randn(1));


        % The whole population is displaced, including the same type of
        % error.
        for i=1:NP
```

```
            population(i,2)=
population(i,2)+disp(1)*(1+MOTION_ERROR*randn(1)); %+err_pos*randn(1);
            population(i,3)=
population(i,3)+disp(2)*(1+MOTION_ERROR*randn(1)); %+err_pos*randn(1);
            population(i,4)=
population(i,4)+disp(3)*(1+MOTION_ERROR*randn(1)); %+err_pos*randn(1);

            % It is checked that the population is inside the map limits.
            for h=2:3
                if population(i,h)<mapmin(h-1)
                    population(i,h)=mapmin(h-1);
                end

                if population(i,h)>mapmax(h-1),
                    population(i,h)=mapmax(h-1);
                end
            end
            if population(i,4)<mapmin(3)
                population(i,4)=population(i,4)+360.0;
            end
            if population(i,4)>mapmax(3)
                population(i,4)=population(i,4)-360.0;
            end

        end

        %  The next option is used for tracking. NP is drastically
reduced.
        if version==6
            if error<50
                NP=10;
            end
        end

        steps=steps+1;

        % A new laser scan is read if the robot is in a new location.
        if dir_disp~='f'

laser_real=dist_est_rob_2D(position,map_real_tr,mapmax,mapmin,err_dis,NUM
_MEASUREMENTS,SENSOR_RES,SENSOR_RANGE,T);
        end

end

% The best solution and the error are returned.
Solution.pose_estimate=bestmem(2:(D+1));
Solution.error=error;

%-------------------------------------------------------------------------
--
% Representation of results

% Obtaining coordinates of redings from true location
observations=zeros(1,NUM_MEASUREMENTS);
```

```
robot=position(1)+1i*position(2);
ang_robot=(position(3)-90)*pi/180;% se pasa a radianes
for index=1:NUM_MEASUREMENTS
    observations(index)=robot+laser_real(index)*exp(1i*(ang_robot+(index-
1)*SENSOR_RES));
end
% Obtaining coordinates of redings from best estimate
est_observations=zeros(1,NUM_MEASUREMENTS);
est_robot=bestmem(2)+1i*bestmem(3);
est_ang_robot=(bestmem(4)-90)*pi/180;% se pasa a radianes
for index=1:NUM_MEASUREMENTS

est_observations(index)=est_robot+laser_real(index)*exp(1i*(est_ang_robot
+(index-1)*SENSOR_RES));
end

% Display robot's position
figure(1)
plot(position(1),position(2),'b+','MarkerSize',5);
% Display observations from true location
for i=1:NUM_MEASUREMENTS

plot(real(observations(i)),imag(observations(i)),'m.','MarkerSize',5);
end
% Display population set
figure(2)
plot(population(:,2),population(:,3),'r.','MarkerSize',5);
% Display observations from best estimate
for i=1:NUM_MEASUREMENTS

plot(real(est_observations(i)),imag(est_observations(i)),'m.','MarkerSize
',5);
end

end
```